UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

TRABAJO FIN DE GRADO

Aplicación de telefonía móvil para discapacitados

Autora: Cristina Archilla Manso

Tutor: Alejandro Sierra Urrecho

Febrero 2014

Resumen

En la sociedad en la que vivimos, no concebimos una vida sin teléfono móvil. En el metro, autobús, en la sala de espera del médico, siempre vemos a alguien utilizando un *smartphone* o teléfono inteligente.

Android, el sistema operativo para dispositivos móviles que abarca más terminales (Samsung, LG, HTC, Sony, etc.), es el más utilizado de todos. Por ello, lo más probable es que veamos a alguien utilizando un teléfono con este sistema operativo. Actualmente, este sistema operativo tiene una gran cantidad de aplicaciones disponibles para su descarga y uso, las cuales hacen de Android una herramienta útil en muchas tareas, desde consultar el correo, jugar y navegar hasta llamar o enviar mensajes de texto. El *smartphone* nos permite tener una comunicación diaria con todo lo que nos rodea: consultar la predicción meteorológica, leer un libro o el periódico, solicitar un taxi, etc.

Pero existe un problema cuando se trata de personas mayores que utilizan un *smartphone*. Este colectivo carece de la destreza necesaria a la hora de manejar estos dispositivos, lo que provoca su exclusión social, ya que en muchos casos son incapaces de abrir un mensaje de texto o, simplemente, encender el teléfono.

Por ello, este trabajo de fin de grado consiste en desarrollar una aplicación que permita a este colectivo utilizar un *smartphone* de última generación como si se tratara de un teléfono antiguo, es decir, una aplicación lista para pulsar un botón y efectuar una llamada de la forma más sencilla y amigable posible.

Palabras clave

Android, teléfono inteligente, dispositivo táctil, aplicación, personas mayores o con dificultades.

Abstract

In our society, we cannot imagine life any more without using cell phones. We use them on the underground, on the bus, at the doctor's waiting room, we can always see somebody using a smartphone.

Android is the most used mobile operating system because it is used by terminals from Samsung, LG, HTC or Sony. Therefore, we are most likely to see somebody using this operating system. Now, Android has a lot of applications available to download. These applications make Android a very useful tool for many tasks: we can check our e-mail, play games, browse the internet, make telephone calls or send text messages to our friends. The smartphone allows us to continuously communicate with our surroundings: we can check the weather forecast, read a book or a newspaper, order a taxi, etc.

But there is a problem when it comes to older people. These people have not enough skills to use these new devices, which causes social exclusion. Unfortunately, too often, they are not able to read a text message or, simply, to turn the terminal on.

This is why the aim of this project is to develop an application that allows the elderly and people with disabilities, to use a smartphone as an old-fashioned cell phone, i.e., an application ready to push a button and make a phone call in the simplest way possible.

Keywords

Android, Smartphone, touch device, application, older people or disabled person.

Agradecimientos

En primer lugar, querría darle las gracias a mi tutor, por darme la oportunidad de realizar este trabajo y así aprender muchísimas cosas de Android que de otra manera no hubiera conocido.

También a todos aquellos compañeros y compañeras de carrera, que han hecho que la carrera haya sido mucho más fácil de digerir, con exámenes de otros años, apuntes... Gracias a todos, ya sabéis quiénes sois.

A mi querida vecina y amiga, por esos días de risas incontroladas y frustraciones compartidas.

A mi chico, que siempre me ha apoyado en todo, ayudándome en lo que podía y con todo el cariño del mundo.

Y finalmente, a mi madre, por estar ahí siempre, desde el principio de todo.

No se puede decir con palabras lo que se siente en lo más profundo. Gracias a todos.

Cristina Archilla

Enero 2014

Índice general

Índice de figuras

Índice de tablas

Glosario

Actividad: cada una de las pantallas de una aplicación Android.

Vista: cada uno de los elementos gráficos de la interfaz.

Contenedor: elemento que almacena y ordena otros elementos como las vistas o, incluso, otros contenedores.

Fragmento: subactividad que permite reutilizar el código y hacer la interfaz más modular.

Preferencia: el mecanismo de almacenamiento más sencillo de Android en forma de pares clave-valor.

API: Sigla de Interfaz de Programación de Aplicaciones, es un conjunto de funciones, métodos o procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

Recurso: datos (cadenas, arrays, imágenes, etc.) que se almacenan independientemente del código de la aplicación para mejorar su organización y permitir su adaptación a cambios de idioma, orientación, etc.

Fichero de manifiesto: fichero que contiene las especificaciones de la aplicación como, por ejemplo, el listado de actividades, los permisos necesarios, etc.

Intención: elemento de comunicación entre los distintos componentes como actividades, servicios, etc.

Fichero de diseño: fichero escrito en XML que especifica la interfaz gráfica de una actividad.

Menú: recurso que sirve para que el usuario pueda seleccionar un elemento de la lista de valores que contiene.

Escuchador: interfaz de la clase `view` que contiene un método `callback` que ha de ser sobrescrito y registrado.

Ciclo de vida: las actividades, a lo largo de su ejecución, pasan por una serie de estados que conllevan la ejecución de una serie de métodos como `onCreate`, `onResume`, `onPause`, etc.

Proveedores de contenidos: especie de base de datos que permite compartir información entre distintas aplicaciones.

1 Introducción

1.1. Motivación

Cada vez es más frecuente el uso de los teléfonos inteligentes o *smarthphones*.

La sociedad ha pasado de utilizar móviles que sólo servían para llamar y mandar mensajes a *smartphones,* con los cuales llaman, juegan, chatean, etc.

Desde que se lanzó el primero en el año 1992, el *IBM Simon*, no se ha parado de innovar en estos dispositivos, pero hasta 2002 no se considera el verdadero nacimiento de los *smartphones*, con el lanzamiento de las PDA's con ranura para tarjetas SIM y la primera *Blackberry*. En el año 2007, Apple lanza el iPhone, y Google el sistema operativo Android. Desde entonces, los *smartphones* tienen cada vez más penetración de mercado: en el año 2008 el 12% de los teléfonos eran *smartphones*, y en el 2013 por primera vez, se vendieron más *smartphones* que teléfonos convencionales.

1.2. Objetivos

El principal objetivo de este proyecto es crear una aplicación que permita a las personas mayores o con dificultades manejar los nuevos *smartphones*, utilizar estos nuevos dispositivos de una manera fácil, cómoda y rápida sin necesidad de tener conocimientos previos para su manejo.

Este proyecto se ha realizado en el sistema Android en lugar de otros como IOS, *WindowsPhone* o *Symbian*, debido a que actualmente Android tiene una cuota de mercado que supera ampliamente a la competencia. Esto hace que sea la plataforma con más aplicaciones, ya que existen más de 187 millones de dispositivos Android frente a los 31,2 millones de dispositivos IOS, el segundo sistema operativo más utilizado.

Dentro del sistema operativo Android, existen varias versiones, ya que el sistema se actualiza frecuentemente para incorporar cambios, nuevas acciones, o para corregir

errores. Por ello, esta aplicación estará disponible para versiones iguales o superiores a la 4.0. Si sumamos el porcentaje de uso de estas versiones obtendremos que la aplicación estará disponible para el 77,4% de los dispositivos Android (el porcentaje de versiones se trata en el capítulo 2).

1.3. Aplicaciones relacionadas

La idea de esta aplicación vino de "Age UK MyPhone" (ver figura 1.1), el cual es un pequeño dispositivo que sirve para realizar llamadas pulsando un nombre en la pantalla. La sencillez de este dispositivo se podía plasmar en una aplicación Android que además pudiera controlar el volumen, el tono de llamada, cambiar el fondo de la pantalla y personalizar la lista de contactos a ver en la pantalla principal.

Figura 1.1. Age UK MyPhone

Para la realización de este proyecto, se han investigado otras aplicaciones similares en Google Play Store.

La primera de ellas ha sido EasyCall (figura 1.2), la cual permite añadir tres números de teléfono manualmente y llamarlos (figura 1.3). Una vez tienes un contacto con un número de teléfono y pulsas sobre dicho número, pide una confirmación para llamarlo.

Figura 1.2. Icono de EasyCall

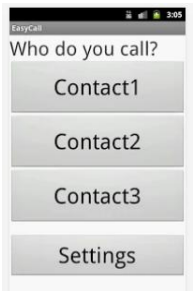


Figura 1.3. Listado de teléfonos

Otra aplicación analizada ha sido Llamar fácil Widget (figura 1.4), la cual te permite seleccionar un contacto de la lista telefónica y crea un acceso directo de ese contacto en la pantalla principal del teléfono. Tras instalarla en un dispositivo con la última versión de Android, no funciona, ya que su última actualización fue el 1 de julio del 2012.

Figura 1.4.Icono de Llamar fácil Widget

La última aplicación analizada ha sido A Simple Call Log (figura 1.5). Esta aplicación permite al usuario ver las llamadas realizadas, recibidas, perdidas y totales (figura 1.6). Por ello, consideramos que es la que más se parece a nuestra aplicación, ya que si pulsamos sobre cualquiera de los números, se llama automáticamente al contacto seleccionado. Tiene *scroll* y también pueden verse el número de llamadas y tiempo hablado por contacto. El menú permite ver el tiempo total llamado, la lista de contactos y otras funciones como ajustes para editar la aplicación.

Figura 1.5. Icono de A Simple Call Log

Figura 1.6. Listado de teléfonos

1.4. Estructura del documento

Para la realización de este proyecto se han tenido en cuenta las fases de desarrollo del software, las cuales se tratan en cada uno de los capítulos de este documento.

En el capítulo 2 se aborda el tema del nacimiento de Android, junto a su historial de versiones y los cambios y novedades que ha tenido cada una de ellas. También se documenta su estructura, desde el núcleo hasta las capas superiores. Seguidamente se comenta la evolución de mercado que ha tenido y lo que se espera que ocurra en el futuro, y finalmente se explican los principales elementos del API utilizados en el proyecto.

En el tercer capítulo se exponen los requisitos funcionales y no funcionales.

En el cuarto capítulo se analiza el diseño utilizado para la realización del proyecto, la estructura de datos utilizada para su realización, el algoritmo utilizado y el diseño exhaustivo de la interfaz.

En el quinto capítulo se analiza la implementación del proyecto, qué componentes del API se han utilizado y cómo.

En el sexto capítulo se comentan varias de las pruebas que se han realizado, divididas en pruebas unitarias, de integración, de sistema o de validación, realizadas con algunos usuarios externos a la aplicación.

El séptimo capítulo está dedicado a las conclusiones que se obtienen de este proyecto.

El octavo capítulo se ocupa de comentar el trabajo futuro que podría tener la aplicación.

2 Estado del arte

2.1. Android

Android es un sistema operativo diseñado inicialmente para teléfonos móviles, basado en Linux, creado con la filosofía del *Opensource*. Estas características hacen que Android sea un sistema operativo multiplataforma, libre y gratuito.

Con el paso del tiempo, Android se ha expandido en los dispositivos táctiles como *smartphones, tablets, phablets, Google* Glasses y varias consolas que han incorporado este sistema operativo.

Android nació orientado a ser multiplataforma, a través del cual muchos terminales utilizan el mismo sistema operativo. Esto fue algo realmente novedoso, ya que hasta ahora se asociaba un sistema operativo a un móvil determinado, y los usuarios no estaban acostumbrados a esto, pero al poco tiempo dio sus frutos, haciendo de Android el sistema operativo más utilizado entre todo tipo de usuarios.

2.1.1 Estructura de Android

La estructura de Android está compuesta por varias capas, como puede apreciarse en la figura 2.1.

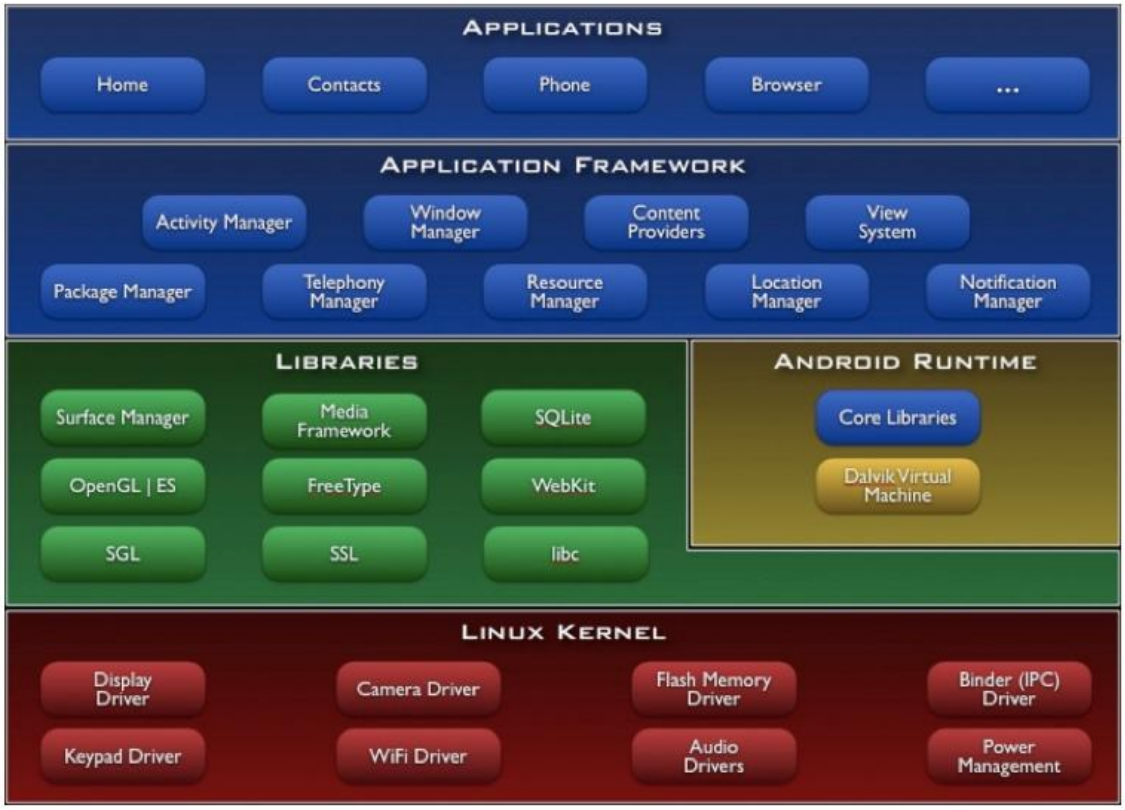


Figura 2.1. Estructura del sistema operativo Android, extraída de Wikipedia bajo licencia de CreativeCommons

El núcleo está basado en el *kernel* de Linux, versión 2.6. Esta capa proporciona servicios como el manejo de memoria, multiproceso, la seguridad, protocolos, etc. El *kernel* es el encargado de la comunicación entre el hardware y el resto de las capas de la arquitectura, esto implica que es dependiente del hardware y para cada elemento del mismo, existe un driver asociado.

Las bibliotecas son las nativas de Android, escritas en C y C++ y están compiladas en código nativo del procesador. La mayoría de estas bibliotecas utilizan código abierto y las más comunes son *WebKit* (para el uso del navegador), *SQLite* (motor de bases de datos), SSL (proporciona servicios de encriptación *Secure Socket Layer*) y *OpenGL* (motor gráfico), entre otras.

El *AndroidRuntime* (el entorno de ejecución) es la capa que está compuesta por las bibliotecas y la máquina virtual de Android. La máquina virtual se llama *Dalvik*, que es una variación de la máquina virtual de Java, creada para optimizar el consumo de energía y rendimiento. *Dalvik* utiliza un *bytecode* distinto al de Java, por ello solo se utiliza Java como lenguaje de programación y los ejecutables llevan la extensión `.dex`(específica de *Dalvik*).

La capa de *ApplicationFrameworkes* el conjunto de APIs que utilizan todas las aplicaciones desarrolladas tanto por Google, por terceros o por usuarios para poder acceder a su funcionalidad. La arquitectura de esta capa está diseñada para que los componentes puedan ser reutilizados. Algunos de los APIs más importantes son *Telephone Manager*, *Activity Manager* (pila de aplicaciones y ciclo de vida de una aplicación), *Content Provider* (encapsula los datos que comparten varias aplicaciones) y *Location Manager* (permite a las aplicaciones obtener la localización actual), entre otras.

Por último, la capa *Applications* está formada por todas las aplicaciones del dispositivo, ya sean nativas, administradas, preinstaladas o instaladas por el usuario. Estas aplicaciones utilizan el API mencionada anteriormente.

2.1.2. Origen de Android

En Octubre de 2003 se funda la empresa Android Inc. en California, Estados Unidos, por Andy Rubin, RichMiner, Nick Sears y Chris White. En agosto del año 2005 Google compra la empresa y contrata a los fundadores para el desarrollo del sistema operativo y, dos años después, se presenta Android como sistema operativo de móviles junto con la Open Handset Alliance, un consorcio de fabricantes de telefonía, para que los nuevos dispositivos móviles utilizaran este reciente sistema operativo.

El 23 de septiembre de 2008 Google lanza oficialmente el SDK de Android 1.0, la primera versión estable del sistema operativo y, un mes más tarde, el 22 de octubre, sale a la venta el primer terminal con dicho sistema operativo, el *HTC Dream G1*.

En esta primera versión nace *AndroidMartket*, el mercado de aplicaciones de Google, en el que se utilizan elementos destacados como el navegador, el soporte para cámara, *googlemaps*, *googletalk* y el reproductor de medios.

2.2. Versiones del S.O.

Tras el lanzamiento de la primera versión estable, Android no ha parado de evolucionar y de lanzar actualizaciones para mejorar el sistema operativo.

El 9 de febrero de 2009 Google difunde la primera actualización del sistema operativo, Android 1.1, de forma exclusiva para el dispositivo *HTC Dream*, que resolvió fallos de la versión anterior, cambió de API (API 2) y añadió algunas

funcionalidades nuevas, como por ejemplo, la posibilidad de guardar adjuntos en los mensajes.

Dos meses después, el 30 de abril, se lanza la versión Android 1.5 (*Cupcake*), que se basa en el núcleo 2.6.27 de Linux y corresponde a la versión API 3. Esta versión incluye algunas novedades como la incorporación de que un contacto pueda tener una foto asociada, la reproducción o grabación en formato MPEG-4 y 3GP y transiciones animadas.

En ese mismo año, el 15 de septiembre se lanza una nueva versión, la 1.6 llamada *Donut*. Corresponde a la versión API 4 y se basa en el núcleo 2.6.29 de Linux. Entre las mejoras de esta versión se encuentran una mejor integración de la cámara, galería y videocámara, siendo su acceso más rápido. También se incluye un motor multilenguaje de síntesis de habla para que las aplicaciones de Android puedan "pronunciar" una cadena de texto.

Las siguientes versiones, 2.0 y 2.1, se llamaron *Eclair*, y se lanzaron el 26 de octubre del 2009 y el 10 de enero de 2010 respectivamente. La versión 2.0 tiene API 5 y la 2.1 tiene API 7. La subversión 2.0.1 cuenta con API 6, que arregla algunos errores de la versión anterior. *Eclair* introduce novedades como la incorporación de mejoras para la cámara (flash, zoom, efecto de colores, etc.) o soporte para bluetooth 2.1. Se mejoran otros aspectos como *Google Maps* 3.1.2 o la optimización del hardware con la GUI renovada. La versión 2.1 realiza pequeños cambios en el API y arregla errores de la versión 2.0.

El 20 de mayo de 2010 se lanza la versión 2.2 llamada *Froyo*, y a lo largo de este año se lanzarán las distintas subversiones (2.2.x) que solucionarán errores y añadirán actualizaciones de seguridad y pequeñas mejoras. *Froyo* se basa en el núcleo 2.6.32 de Linux y se corresponde a la versión API 8. Entre las novedades de esta versión se incluyen el soporte para Adobe Flash, soporte para contraseñas alfanuméricas, optimización de memoria y rendimiento, opción de deshabilitar el acceso de datos móviles, etc.

A finales de 2010, el 6 de diciembre, Google lanza la versión 2.3, llamada *Gingerbread*, la cual se basa en el núcleo 2.6.35 de Linux. El resto de subversiones fueron lanzadas a lo largo del año 2011, hasta la última (2.3.7), el 21 de septiembre.

Gingerbread cuenta con dos versiones de API: la versión 9 corresponde a la versión y subversiones Android 2.3, 2.3.1 y 2.3.2, mientras que la versión API 10 (lanzada el 9 de febrero) corresponde a las subversiones restantes (2.3.3, 2.3.4, 2.3.5, 2.3.6 y 2.3.7).

La versión 2.3 incluye novedades como un recolector de basuras para incrementar el rendimiento, mejoras en audio, gráficos orientados a juegos, soporte para pantallas extra grandes, entrada de texto virtual mejorada(más rápida y precisa), actualización de la interfaz para mejorar la velocidad de procesamiento, etc.

Las versiones que componen el API 10 subsanan algunos errores del anterior API, mejoran la eficiencia de la batería, el rendimiento del sistema y algunos errores que eran propios de algunos terminales (como el bluetooth en el Samsung Galaxy S).

Tras el nacimiento de las *tablets*, Google expande su mercado y como consecuencia de esto, el 22 de febrero de 2011, nace la versión Android 3.0 (*Honeycomb*), pensada para funcionar en *tablets* (excluyendo a los *smartphones*). El núcleo que utiliza se basa en la versión 2.6.36 de Linux y el primer terminal en instalarla fue la *Tablet Xoom* de Motorola, que se puso a la venta dos días después del lanzamiento de *Honeycomb*.

La versión 3.0 corresponde a la versión de API 11, y, al igual que pasaba en *Gingerbread*, *Honeycomb* tiene varias API. Para la versión 3.1 el API 12 y para la 3.2 el número 13.

Entre las novedades de *Honeycomb* se encuentran una nueva interfaz optimizada para *tablets*, se crea la barra de acción, se rediseña el teclado, optimizándolo para pantallas grandes, se mejora el uso de HTTPS, etc.

Las versiones 3.1 y 3.2 arreglan fallos de la versión 3.0 y añaden nuevas funciones como la conectividad de accesorios USB, soporte de joysticks, *gamepads*, teclados externos y punteros, etc.

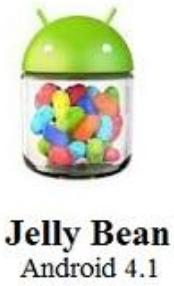
Tras la incorporación de las *tablets* al mercado y el nacimiento de nuevos *smartphones* de pantallas grandes, Google lanza una nueva versión de Android, la 4.0

llamada *Ice CreamSandwich* el 19 de octubre de 2011, que engloba todos los conceptos nuevos que se han utilizado en las *tablets*, ahora orientados a todos los dispositivos táctiles. *Ice CreamSandwich* se basa en el núcleo 3.0.1 de Linux.

Esta versión, junto con las subversiones 4.0.1 y 4.0.2, corresponde al API 14 y las subversiones 4.0.3 y 4.0.4 al API 15.

Ice CreamSandwich trae varias novedades como la posibilidad de acceder a las aplicaciones desde la pantalla de teléfono bloqueado, se añade una nueva tipografía (*Roboto*), la posibilidad de grabar videos en alta definición, desbloqueo facial, la aceleración de la interfaz mediante hardware, etc.

Las subversiones arreglan errores, mejoran el rendimiento de la cámara, hacen que la rotación de la pantalla sea más fluida, se mejoran los gráficos, se optimizan las bases de datos, se mejora el calendario y las funcionalidades del *bluetooth*.



El 9 de julio de 2012 se lanza la versión 4.1, *JellyBean*, que corresponde a la versión de API 16 y se basa en la versión de núcleo 3.0.31 de Linux.

En esta versión se implementa una interfaz de usuario más fluida, aumentando la velocidad de *fps* a 60, se utiliza un triple buffer en los pipelines y se extiende la latencia *vsync*.

La versión 4.2, lanzada el 29 de octubre del 2012, mantiene el mismo nombre que la anterior, *JellyBean*, pero la versión de API es 17. Sus novedades son que se actualizan los diccionarios, se incluye soporte multiusuario y se mejora el teclado con escritura gestual (que permite escribir utilizando gestos sobre el teclado QWERTY), etc.

Por último, existe otra versión llamada *JellyBean*, que corresponde a la 4.3 (desde el 24 de julio de 2012), cuya versión API es 18 y que se vendió inicialmente en el *Nexus 7*. Las principales características son que mejora la seguridad, se localiza *wifi* en segundo plano, se mejora la escritura y se soporta el *bluetooth* con poca batería.

Finalmente, el 1 de noviembre de 2013, Google lanza su última versión de Android hasta la fecha, conocida como *Kit-kat*. Su versión del *API* es la 19 e inicialmente se vendió únicamente sobre el dispositivo *Nexus 5*.

Las principales características de esta última versión son que se agilizan los componentes para tratar de reducir el consumo de batería, el almacenamiento está integrado en la nube, para las llamadas de números que no se encuentran en la agenda realiza una búsqueda de coincidencias de empresas con una lista local de *Google Maps*, etc.

Actualmente conviven varias de las versiones de Android, dado que muchos terminales tienen instaladas versiones antiguas y no se han actualizado a las nuevas. En el siguiente gráfico se puede apreciar el porcentaje de terminales que utilizan las distintas versiones del sistema operativo (a excepción de aquellas utilizadas por menos de un 0,1%, que no se muestran).

Versión	Codename	API	Distribution
2.2	Froyo	8	1,3%
2.3.3 2.3.7	Gingerbread	10	21,2%
3.2	Honeycomb	13	0,1%
4.0.3 4.0.4	Ice CreamSandwich	15	16,9%
4.1.x	JellyBean	16	35,9%
4.2.x		17	15,4%
4.3		18	7,8%
4.4	Kit-Kat	19	1,4%

Tabla 2.1 con la distribución de las versiones de Android a día 8 de Enero de 2014 extraída de developers.android

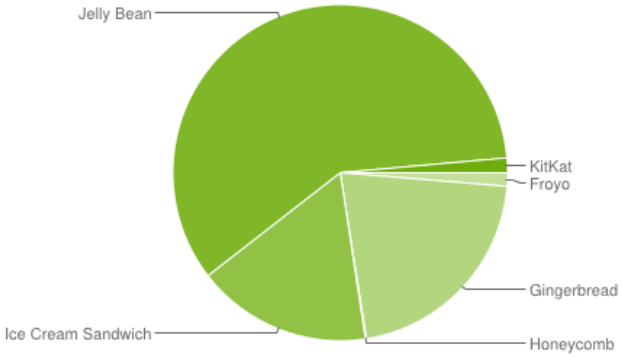


Figura 2.2 con la distribución de las versiones de Android a día 8 de Enero de 2014 extraído de developers.android.

Como se puede observar tanto en la tabla 2.1 y la figura 2.2, un cuarto de los usuarios siguen utilizando la versión *Gingerbread*, aunque cada vez más usuarios utilizan las últimas versiones, ya que más de la mitad del mercado emplea alguna de las versiones de *JellyBean*, y *Kit-Kat* cada vez tiene más usuarios.

Esto se debe a que existe un mercado enorme de teléfonos Android y los desarrolladores no pueden dar soporte de todas las versiones en todos los dispositivos; por ello, muchos de los terminales continúan utilizando versiones antiguas.

2.3. Evolución del mercado y pronóstico de futuro

Desde el nacimiento de Android, su número de ventas ha ido creciendo de modo exponencial, tanto en España como en el resto de países. El primer terminal Android se vendió en Estados Unidos en el año 2008, pero hasta 2009 no llegó a Europa, lo que hizo que llegaran las versiones 1.0 y 1.1 simultáneamente.

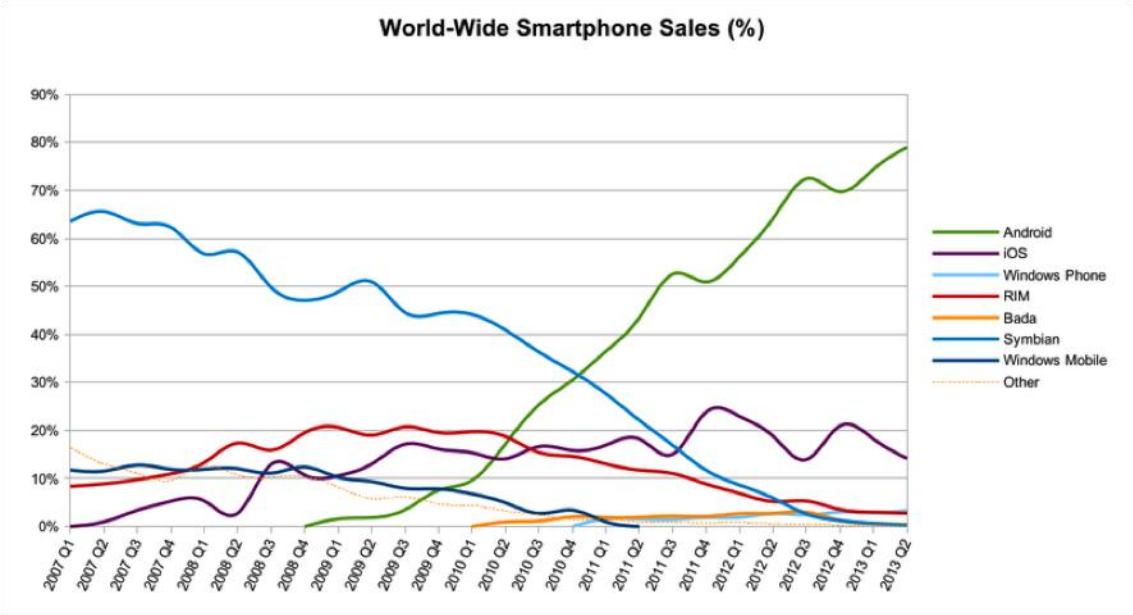


Figura 2.3.1 Gráfico de evolución del mercado extraído de Wikipedia, bajo licencia de CreativeCommons

Como se puede observar en la figura 2.3.1, desde que se lanzó Android en 2009 su crecimiento ha superado a toda la competencia, y se proclama líder a nivel mundial de los dispositivos móviles en el cuarto trimestre de 2010 (con las versiones de *Gingerbread*).

Este mismo panorama se ha repetido prácticamente en todos los países industrializados, a excepción de algunos como Japón, donde continúa ganando IOS.

Germany	3 m/e Oct 2012	3 m/e Oct 2013	% pt. Change
Android	75.6	77.5	1.9
BlackBerry	2.0	0.7	-1.3
iOS	15.4	13.8	-1.6
Windows	2.3	6.3	4.0
Other	4.7	1.6	-3.1
GB	**3 m/e Oct 2012**	**3 m/e Oct 2013**	**% pt. Change**
Android	54.0	55.6	1.6
BlackBerry	8.1	3.3	-4.8
iOS	32.7	28.7	-4.0
Windows	4.6	11.9	7.3
Other	0.5	0.5	0.0
France	**3 m/e Oct 2012**	**3 m/e Oct 2013**	**% pt. Change**
Android	60.4	68.1	7.7
BlackBerry	6.4	2.6	-3.8
iOS	19.5	15.9	-3.6
Windows	5.1	12.5	7.4
Other	8.5	0.9	-7.6
Italy	**3 m/e Oct 2012**	**3 m/e Oct 2013**	**% pt. Change**
Android	57.3	68.8	11.5
BlackBerry	4.0	1.4	-2.6
iOS	18.5	10.1	-8.4
Windows	11.7	16.1	4.4
Other	8.5	3.5	-5.0
Spain	**3 m/e Oct 2012**	**3 m/e Oct 2013**	**% pt. Change**
Android	84.0	90.1	6.1
BlackBerry	3.4	0.0	-3.4
iOS	3.2	4.3	1.1
Windows	2.3	4.3	2.0
Other	7.2	1.3	-5.9

USA	3 m/e Oct 2012	3 m/e Oct 2013	% pt. Change
Android	47.7	52.6	4.9
BlackBerry	1.6	0.8	-0.8
iOS	47.2	40.8	-6.4
Windows	2.5	4.8	2.3
Other	0.9	1.0	0.1
China	**3 m/e Oct 2012**	**3 m/e Oct 2013**	**% pt. Change**
Android	69.7	78.1	8.4
BlackBerry	0.1	0.0	-0.1
iOS	18.9	15.5	-3.4
Windows	3.9	3.5	-0.4
Other	7.4	2.8	-4.6
Australia	**3 m/e Oct 2012**	**3 m/e Oct 2013**	**% pt. Change**
Android	60.4	54.9	-5.5
BlackBerry	0.8	1.6	0.8
iOS	32.8	35.0	2.2
Windows	4.0	7.3	3.3
Other	2.0	1.1	-0.9
Japan	**3 m/e Oct 2012**	**3 m/e Oct 2013**	**% pt. Change**
Android	N/A	36.2	N/A
BlackBerry		0.3	
iOS		61.1	
Windows		0.4	
Other		1.9	
EU5	**3 m/e Oct 2012**	**3 m/e Oct 2013**	**% pt. Change**
Android	64.5	70.9	6.4
BlackBerry	5.1	1.8	-3.4
iOS	20.8	15.8	-5.0
Windows	4.8	10.2	5.4
Other	4.9	1.4	-3.4

Figura 2.3.2 Tabla con la evolución del mercado en varios países en el periodo desde octubre 2012 hasta octubre de 2013 bajo licencia de http://www.kantarworldpanel.com

Como podemos observar en la figura 2.3.2, en prácticamente todos los países se ha incrementado la cuota de Android; concretamente en España supera ya el 90%.

A nivel mundial, a mediados del mes de noviembre de 2013 se supera en un 81% la cuota de mercado de Android, con más de 211 millones de terminales, un 40% más que el año anterior.

Esta tendencia de aumento de la cuota de mercado podría estar empezando a quedar estancada, ya que en el mes de mayo de 2013 el porcentaje en España para Android era del 92,5%, y en el mes de octubre parece que disminuye al 90,1%. Esto

podría significar que la tendencia a aumentar se va hacer más lentamente, ya que, aunque siga aumentando la cuota (en el último año aumentó en 6 puntos), lo hará más despacio.

Finalmente, cabe destacar que la principal empresa que más terminales vende y que utiliza el sistema operativo Android es Samsung, la cual constituye más del 60% de terminales con Android, como podemos apreciar en la figura 2.3.3.

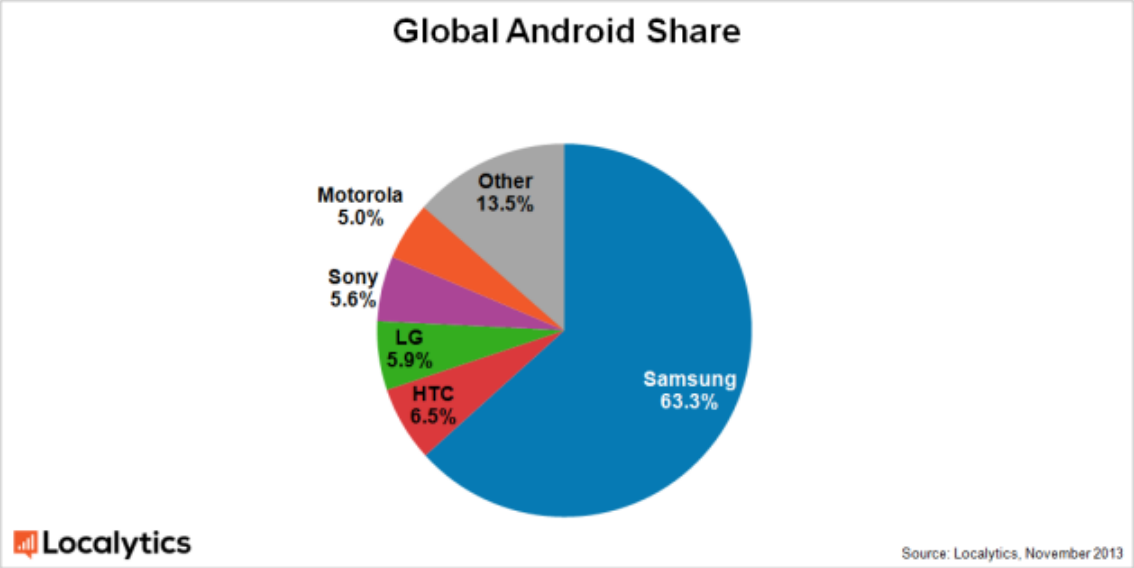


Figura 2.3.3. Gráfico de dispositivos que utilizan Android.

2.4. Conceptos de Android utilizados

Para la realización de esta aplicación, se han utilizado los siguientes elementos:

- Actividad: una aplicación Android es básicamente un conjunto de actividades o pantallas. Cada actividad se implementa mediante un fichero de código Java y opcionalmente, un fichero de diseño XML. Las actividades interactúan con los usuarios a través de su interfaz construida a base de objetos de tipo `View` y `ViewGroup` que normalmente se conocen como vistas y contenedores, respectivamente. Las actividades tienen un ciclo de vida, ya que se crean, se paran, se recargan y se destruyen. En el siguiente gráfico se puede ver dicho ciclo de vida:

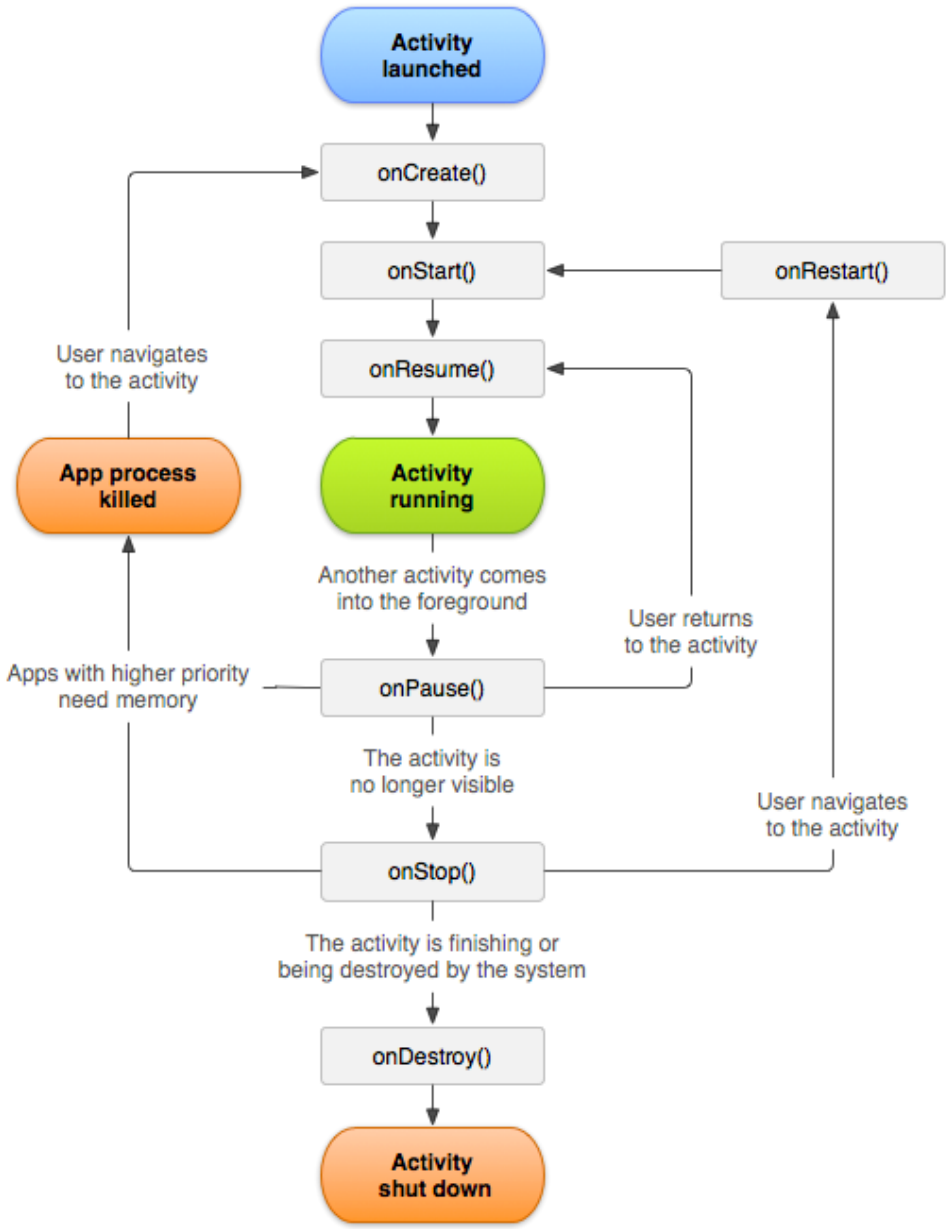


Figura 2.4.1. Ciclo de vida de una actividad extraída de developer.android.com

- Contenedores: elementos gráficos que almacenan y organizan otros elementos como las vistas. Entre los contenedores más utilizados se encuentran `RelativeLayout`, `TableLayout` o `LinearLayout`.

  El ejemplo siguiente muestra un `LinearLayout` en código XML.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayoutxmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:orientation="vertical" />
```

- Vistas: elementos gráficos que se incluyen en las actividades y permiten interactuar al usuario con la aplicación. Algunas de las vistan más utilizadas son los botones y los campos de texto. El siguiente ejemplo muestra un botón en código XML.

```
<Button android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello, I am a Button"/>
```

- Ficheros de diseño: ficheros escritos en el lenguaje XML que se almacenan en la carpeta `res/layout`. Estos ficheros sirven para almacenar las definiciones de los elementos que posteriormente va a utilizar la interfaz de la aplicación. Estos ficheros están asociados a una actividad o fragmento y pueden contener definiciones de animaciones, colores, elementos de interfaz (botones, fondos), menús, cadenas de texto, estilos, etc.

  Un ejemplo de fichero de diseño es el siguiente, que utiliza un contenedor de tipo `RelativeLayout` y una vista de tipo `TextView`:

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity" >
    <TextView
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:layout_centerHorizontal="true"
       android:layout_centerVertical="true"
       android:text="@string/hello_world" />
</RelativeLayout>
```

- Intenciones: (intents en inglés) objetos que sirven para invocar actividades, servicios o proveedores de contenido. Para llamar a una actividad utilizamos una intención como en el siguiente código:

```
Intent intent = new Intent (this, orderPersonalized.class);
startActivity(intent);
```

En el siguiente ejemplo se ve como mediante intenciones se realiza una llamada del teléfono:

```
Intent callIntent = new Intent(Intent.ACTION_CALL, Uri.parse("tel:976123123"));
startActivity(callIntent);
```

- Escuchadores de eventos: interfaz de la clase `View` que contiene un método *callback* que ha de ser sobrescrito y registrado. Existe un método *callback* por cada escuchador de eventos y algunos de escuchadores son los siguientes: `onClick()`, `onFocusChange()`, `onKey()`, `onTouch()` y `onCreateContextMenu()`. Un ejemplo de ello es el siguiente código:

```
Button boton = (Button)findViewById(R.id.boton);
boton.setOnClickListener( new OnClickListener()
        public void onClick(View v) {
                Toast.makeText(MainActivity.this, "Button Clicked",
                        Toast.LENGTH_SHORT).show();
        }
});
```

- Fragmentos: la versión 3.0 de Android, *Honeycomb*, está orientada a las tabletas, por lo que se introdujeron nuevos elementos, entre ellos muchos de interfaz. Los fragmentos son uno de esos nuevos elementos, que se comportan como actividades y tienen su propio ciclo de vida. Estos nuevos elementos se pueden reutilizar en distintas actividades, aunque siempre deben ir ligados a una en concreto. También se pueden emplear en versiones inferiores a la 3.0 con un paquete llamado *SupportPackage*, que se importa como *Android.support.v4*.

  Estos fragmentos permiten modificar dinámicamente la interfaz de cualquier dispositivo, sin la necesidad de cambios complejos. Un ejemplo de esto es la figura 2.4.3, en la que podemos ver que en una tableta se muestran dos fragmentos (A y B) en la misma actividad, y en un *smartphone* se muestran esos mismos fragmentos, pero uno porcada actividad (la pantalla es mucho menor y no se podrían ver los dos simultáneamente).

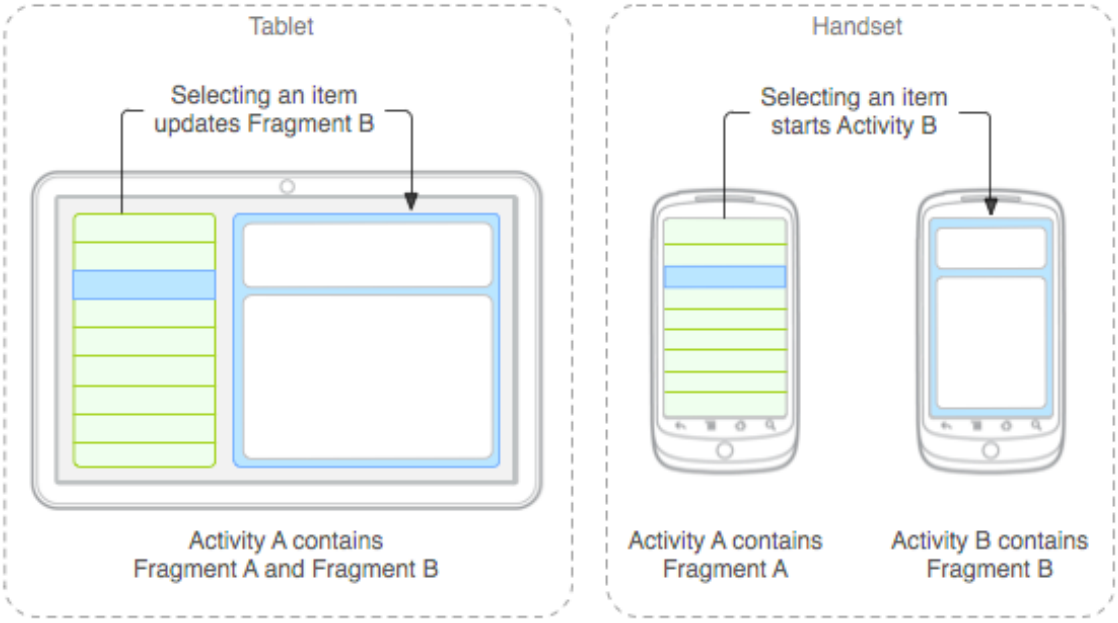


Figura 2.4.3. Fragmentos. Imagen extraída de developer.android.com

- Diálogos: estos elementos permiten interactuar con los usuarios en forma de ventana flotante. Se utilizan principalmente como alertas o para pedir al usuario que responda a una pregunta o elija un elemento de una lista de selección.

  Antes los diálogos se extendían de la clase `Activity`, pero desde la versión 3.0 se extienden de `DialogFragment`, y para las versiones anteriores se utiliza la librería de compatibilidad *Android-support-v4.jar*.

  Un ejemplo de diálogos es la figura 2.4.4.

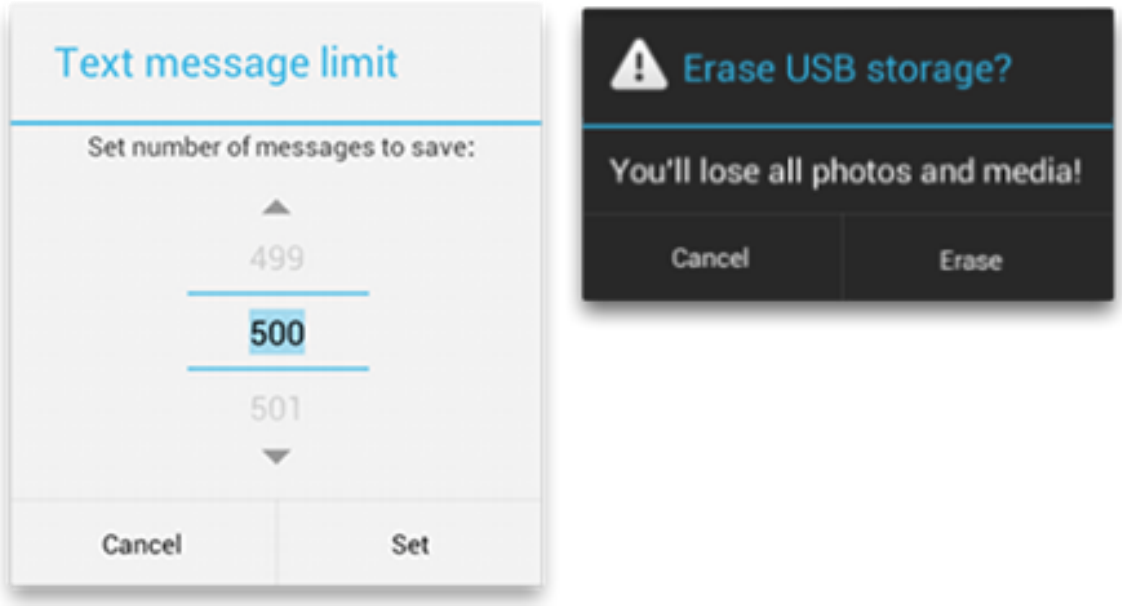


Figura 2.4.4. Diálogos. Imagen de developer.android.com

- Preferencias: las preferencias permiten, al igual que las bases de datos *SQLite*, guardar información. Las preferencias se almacenan utilizando el método clave-valor y, a diferencia de las bases de datos *SQLite*, los datos se guardan en un archivo XML que reside en la carpeta `shared_prefs`. Al igual que los diálogos, antes heredaban de las actividades pero ahora lo hacen de los fragmentos, utilizando `PreferenceFragment`, mediante transacciones para la gestión de los datos. Es frecuente acceder a las preferencias a través del menú de preferencias que puede contener distintos elementos gráficos como *checkbox*, listas, diálogos y campos de texto.

  Un buen ejemplo de preferencias serían los ajustes de cualquier dispositivo Android, como se muestra en la figura 2.4.5:

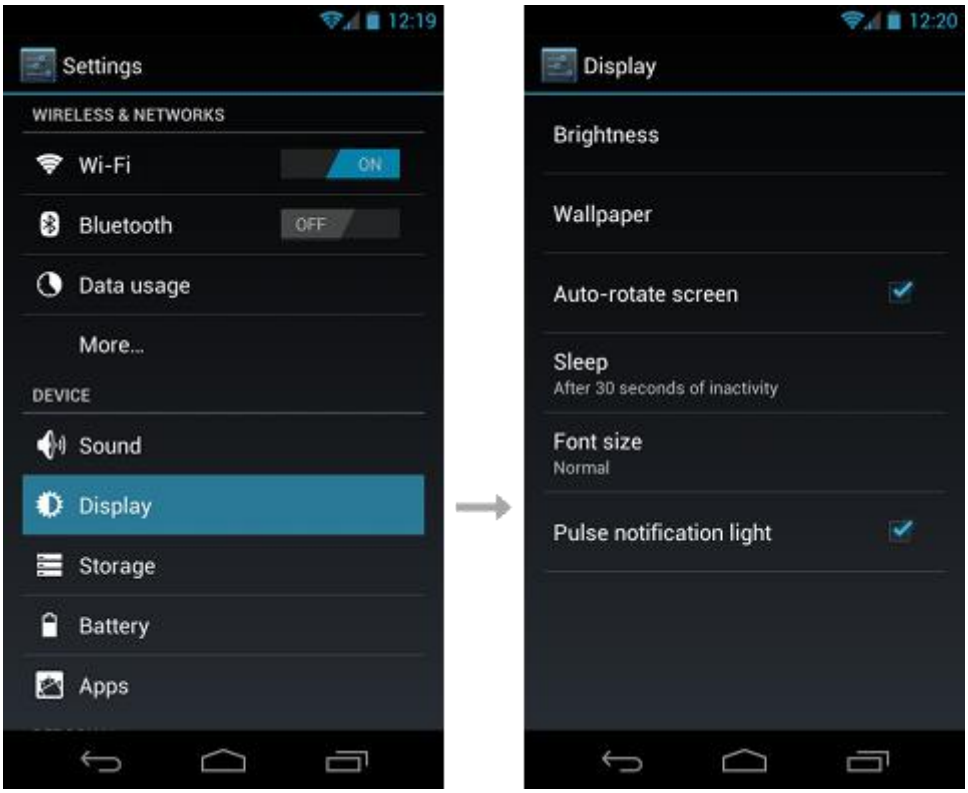


Figura 2.4.5. Preferencias. Imagen de developer.android.com

- Menús: los menús son un elemento que aparece al pulsar el botón "menú" del teléfono y, en las últimas APIs, al pulsar el botón con puntos suspensivos que se ve en la parte derecha de la barra de acción. Tras esto, se despliegan las opciones guardadas en el fichero XML del menú correspondientes, pudiendo seleccionar cualquiera de ellas. A partir de la versión 3.0 los menús suelen arrancarse desde

la barra de acción. Como podemos observar en la figura 2.4.6, en la imagen de la izquierda se utiliza un menú clásico, y en la de la derecha se utiliza uno basado en la barra de acción. La imagen de la izquierda está tomada con un *smartphone* Android con versión 4.0.4, y la de la derecha con un *smartphone* con versión 4.4.

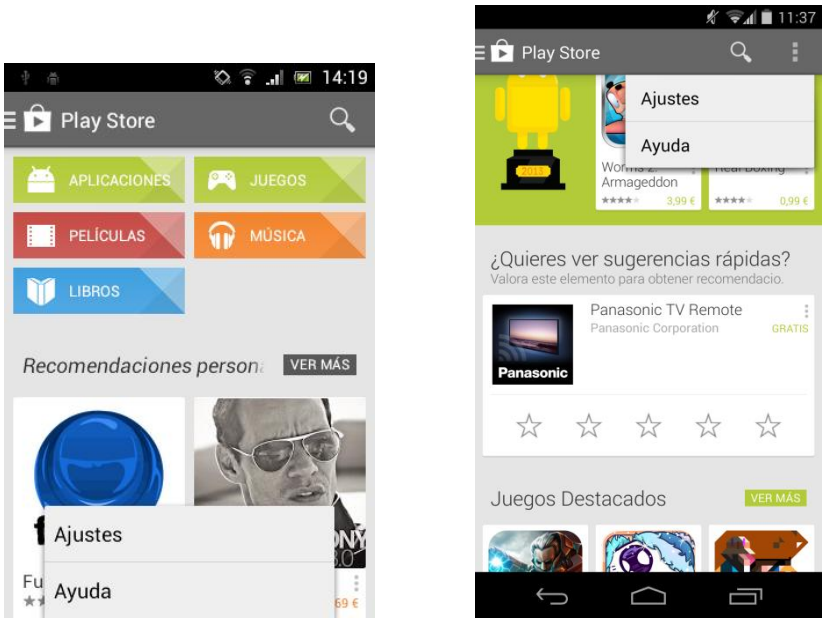


Figura 2.4.6 Menú en Android 4.0.4 y en 4.4.

- Recursos: los recursos en Android se almacenan en la carpeta `/res` dentro de la estructura del proyecto, y pueden ser imágenes, cadenas de texto, animaciones, estilos, etc. Dentro de esta carpeta `/res` se clasifican los recursos por carpetas. Por defecto se crean las siguientes:
  - `drawable`: contiene las imágenes utilizadas por la aplicación y se puede dividir en varias subcarpetas dependiendo de la densidad y tamaño de la pantalla.
  - `layout`: contiene los ficheros XML de cada una de las pantallas de la aplicación.
  - `values`: contiene los ficheros XML de varios tipos de recursos, como cadenas de caracteres(`strings.xml`), estilos (`styles.xml`), colores (`colors.xml`), etc.

Otras carpetas que se pueden añadir son `menu` (para definir los menús de la aplicación, `xml` (otros ficheros que utiliza la aplicación pero no se engloban en ninguna de las carpetas), etc.

- Proveedores de contenidos: permite a las aplicaciones que no tienen relación entre ellas compartir datos que se almacenan en una base de datos SQLite. Para poder acceder a los datos que contiene la base de datos, es necesario conocer los nombres de los campos y las tablas. Algunos de los proveedores de contenidos que tiene Android son CallLog, Contacts o Calendar.

  Para acceder a un proveedor de contenidos concreto, se ha de identificar con la URI. Una URI es una cadena de texto que permite identificar un recurso siguiendo el siguiente patrón:

  ```
  content:// <authority>/<data_path>/<id>
  ```

  Un par de ejemplos de URI son los siguientes:

  ```
  content://call_log/calls
  content://call_log/calls/2
  content://media/internal/images
  ```

  El acceso a las tablas se realiza mediante una consulta a la base de datos. Los datos se extraen con un tipo de dato llamado `Cursor`, el cual tiene almacenada la estructura de la tabla analizada y se aplica sobre una URI determinada.

```
Cursor c= contentResolver.query(
        Uri uri, //Uri determinada
        String[]projection, //Lista de columnas a devolver
        String selection, //Cláusula SQL WHERE
        String[]selectionArgs, //Argumentos de selección
        String sortOrder)//Cláusula SQL ORDER BY
```

3 Catálogo de requisitos

El catálogo de requisitos es la especificación del comportamiento que se espera de cualquier proyecto software. Para elaborar este catálogo en el proyecto FastCall, se han definido los siguientes requisitos, divididos en funcionales y no funcionales.

3.1. Requisitos funcionales

Los requisitos funcionales son aquellas acciones que han de ocurrir en algún momento en el proyecto. Son los siguientes:

RF1:obtener el listado de contactos.

El usuario podrá seleccionar qué tipo de listado desea tener en su actividad principal en la pantalla de ajustes. Las opciones a elegir son:

- Realizadas: los números a los cuales el usuario ha llamado más veces.
- Recibidas: los números que más han llamado al usuario.
- Totales: los números ordenados por la suma de las llamadas realizadas, recibidas y perdidas.
- Personalizado: el usuario accede al contenido de la agenda del teléfono y puede seleccionar los contactos que desea para añadir a su lista personalizada.

RF2:seleccionar el número de contactos a mostrar: el usuario en las preferencias selecciona el número de contactos que desea ver para los siguientes casos: llamadas realizadas, recibidas o totales (desde 1 hasta 8).

RF3:mostrar contactos: se muestra en la ventana principal el número de contactos con el orden seleccionado por el usuario. Si son de 1 a 4 usuarios se visualizan centrados en la pantalla, y en caso de 5 a 8 se dividen en dos filas.

RF4:llamar a los contactos: se puede llamar a cada uno de los contactos mostrados en la ventana principal, pulsando sobre el nombre del contacto.

RF5:cambiar tono de llamada: en la ventana de preferencias, al seleccionar "cambiar el todo de llamada" se permite seleccionar un tono de la lista del móvil.

RF6:modificar volumen: el usuario selecciona el volumen del tono de llamada. Hay 5 valores posibles:

- Silencio: no se escucha nada.
- Vibración: el teléfono vibra pero no suena.
- Tono bajo: el tono suena a una cuarta parte del volumen total.
- Tono medio: el tono suena a la mitad del volumen total.
- Tono alto: el tono suena al volumen máximo

RF7:añadir contacto a la agenda: el usuario puede añadir un nuevo contacto a la agenda del teléfono.

RF8:cambiar el fondo de pantalla: el usuario puede cambiar el fondo de la pantalla principal, pudiendo elegir entre dos galerías distintas:

- Fondos de la aplicación: la aplicación ofrece cuatro fondos mostrados en una galería.
- Galería del móvil: se accede a la galería del móvil para seleccionar una imagen.

3.2. Requisitos no funcionales

Los requisitos no funcionales son los requisitos de interfaz, usabilidad, operacionales, documentación, seguridad, mantenimiento, etc. Para este proyecto, los siguientes:

RNF1:esta aplicación está orientada a personas mayores.

RNF2:los usuarios que manejen esta aplicación carecen de los conocimientos necesarios para manejar un dispositivo móvil de última generación.

RNF3:la aplicación está orientada a dispositivos táctiles con el sistema operativo Android a partir de la versión 4.0.

RNF4: la seguridad de los datos del usuario está garantizada, ya que no se utilizarán los datos de los usuarios fuera de la lógica de la aplicación.

RNF5:no se necesita conectividad de internet salvo para la descarga desde Google Play, por lo que su estabilidad está garantizada.

RNF6:habrá interoperabilidad entre la aplicación y las funcionalidades del *smartphone* (llamadas, acceder a la agenda, etc.).

RNF7:la aplicación funcionará correctamente siempre que el *smartphone* tenga tarjeta SIM para poder realizar las llamadas.

RNF8:la aplicación estará siempre en primer plano para evitar que los usuarios tengan que buscarla y abrirla.

RNF9:la aplicación se autoarrancará al encender el *smartphone*.

RNF10:existirá mantenimiento de la aplicación ante el lanzamiento de nuevas versiones del sistema operativo Android.

RNF11:el idioma de la aplicación se adaptará al idioma del teléfono en caso de que este esté en inglés.

RNF12:la interfaz se adapta a todo tipo de pantallas, tanto grandes como pequeñas.

4 Diseño de la solución

Para el proyecto FastCall se ha realizado un diseño que abarca todos los requisitos descritos en el apartado anterior. Este diseño proporciona una idea completa de lo que es el software desarrollado. Para ello, el diseño se subdivide en el diseño de la interfaz, estructura de datos y algoritmos.

4.1. Diseño de la interfaz

La interfaz se diseñó con el propósito de ser muy práctica. El diseño de la interfaz está detallado en la imagen 4.1. En ella podemos observar que la actividad principal consta de la lista de teléfonos que el usuario ha elegido para realizar llamadas. Desde esta actividad accederemos mediante un menú a la configuración de la aplicación.

En la configuración de la aplicación se podrá seleccionar el tono y volumen del teléfono para las llamadas entrantes. También se podrá cambiar el fondo de la actividad principal, pudiendo elegir entre cuatro fondos definidos por la aplicación o una imagen de la galería de fotos del dispositivo. Si el usuario quiere cambiar la lista de números que aparece en la pantalla principal, puede elegir si desea ver las llamadas por recibidas, realizadas, totales (que es la suma de recibidas, realizadas y llamadas perdidas) o un orden personalizado. Para los tres primeros se podrá elegir el número de contactos que quiere ver en el ajuste "número", siendo ocho el máximo.

Si el usuario selecciona el orden personalizado, deberá crear la lista de números favoritos seleccionando los contactos de su agenda personal. En caso de que uno de los números que desee añadir no se encontrase en la lista, se podrá agregar desde ese mismo lugar directamente sin perder la lista hasta entonces seleccionada. Por último, al aceptar, se actualizará la página principal con los cambios efectuados.

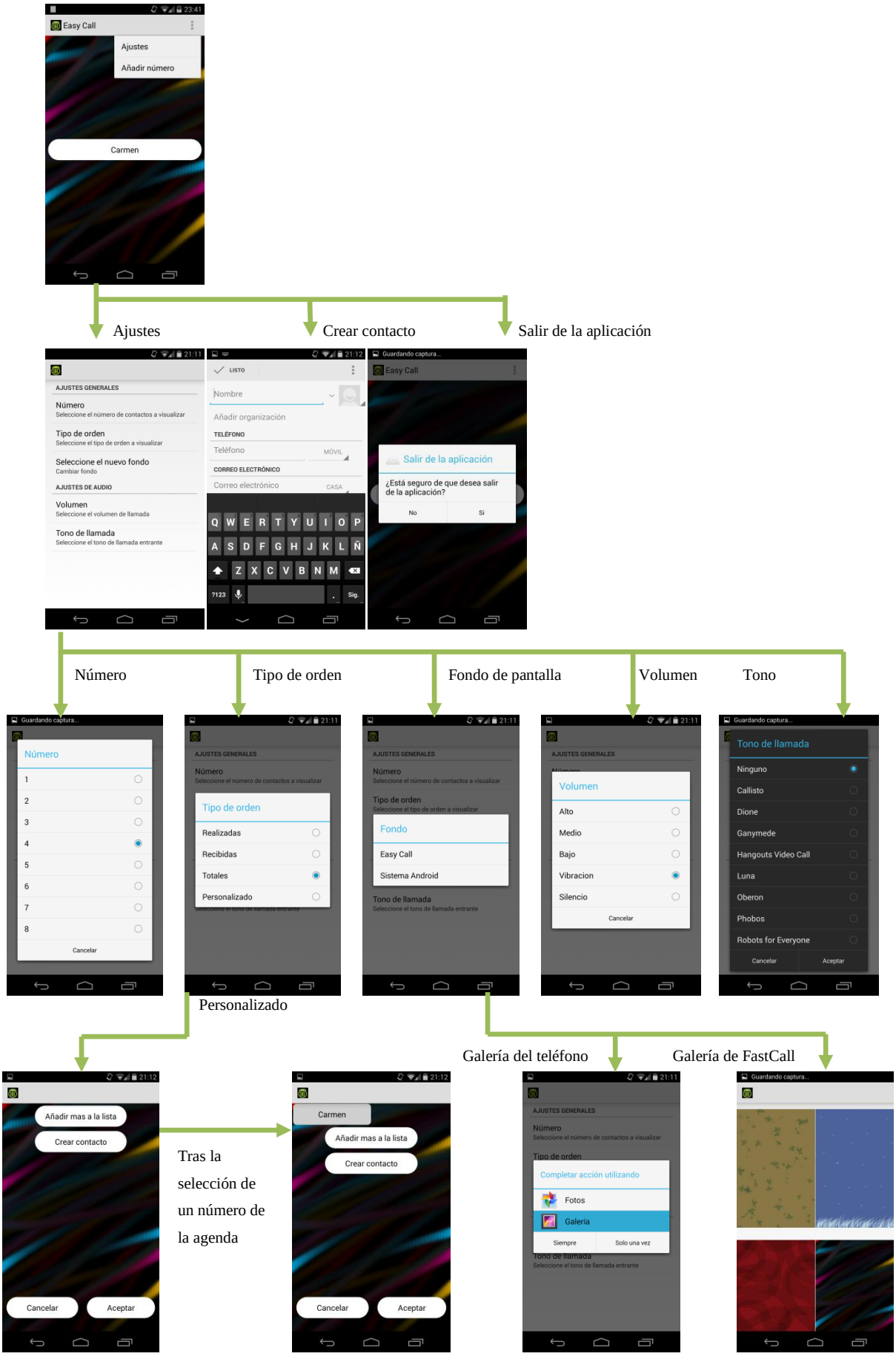


Imagen 4.1 Diseño de la interfaz

4.2. Diseño de la estructura de datos

Para almacenar los datos extraídos de la agenda telefónica y de la lista de llamadas del dispositivo móvil se utiliza una estructura que consta de un campo de texto denominado `nombre`, otro campo de texto denominado `número` y un campo de números enteros, el cual se denomina `contador`.

El campo `contador` se inicializa a uno para cada número insertado y se incrementa por cada vez que el número telefónico sea utilizado dependiendo del criterio seleccionado (llamadas realizadas, recibidas o totales).

Si la lista de llamadas utiliza el criterio personalizado, el campo `contador` quedará inservible dado que no se utilizará, pues no se analizarán las llamadas efectuadas con el dispositivo móvil.

Para evitar resultados duplicados se comprobará que el teléfono o el nombre no estén repetidos; es decir, si existen dos contactos con números diferentes pero con el mismo nombre, se identificará como si fuera el mismo contacto, y se incrementará el contador en lugar de añadir el contacto a la lista de números utilizados.

4.3. Diseño de algoritmos

Para obtener el listado de los números más utilizados (llamadas recibidas, realizadas y totales), se ha utilizado el algoritmo Mergesort, que se basa en la técnica "divide y vencerás".

Este algoritmo, desarrollado en 1945 por John Von Neuman, es estable, y su complejidad es $O(n \log n)$.

En comparación con otros algoritmos de ordenación, Mergesort tiene una complejidad menor que Bubblesort($O(n^2)$) y es más estable que Quicksort, a pesar de que éste último es más rápido.

El pseudocódigo de este algoritmo es el siguiente:

```
function mergesort(array A[x..y])
begin
  if ((y-x) > 0):
    array A1 := mergesort(A[x..(int( x+y / 2))])
    array A2 := mergesort(A[int(1+(x+y / 2))..y])
    return merge(A1, A2)
  else:
    return A
end

function merge(array A1[0..n1], array A2[0..n2])
begin
  integer p1 := 0
  integer p2 := 0
  array R[0..(n1 + n2 + 2)]//suponiendo que n1 y n2 son las posiciones
  //del array y no el length de este mismo, de otro modo seria (n1 + n2)
  while (p1 <= n1 and p2 <= n2):
    if (A1[p1] <= A2[p2]):
      R[p1 + p2] := A1[p1]
      p1 := p1 + 1

    else
      if (A1[p1] > A2[p2]):
        R[p1 + p2] := A2[p2]
        p2 := p2 + 1
  return R
end
```

Figura 4.2. Pseudocódigo del algoritmo Mergesort extraído de Wikipedia.

5 | Implementación

Para el desarrollo de la aplicación FastCall se han utilizado varios de los elementos definidos en el API de Android, por supuesto, en el lenguaje de programación Java. A continuación se describen todos los componentes implementados y su funcionalidad.

5.1. Estructura de datos

Para la realización de la aplicación, se ha creado una clase en Java llamada `contactStruct.java`, que implementa la estructura de datos necesaria para crear la lista de teléfonos seleccionada por el usuario. Esta clase consta de tres valores, un tipo numérico y dos cadenas de texto, como se puede apreciar en el código siguiente:

```
public class contactStruct {

      private String numTel;
      private int cont;
      private String nombre;
      ...
}
```

Esta clase contiene los correspondientes métodos para insertar y extraer valores.

5.2. Actividades

Cada actividad en Android lleva asociada una clase Java que implementa sus métodos. En nuestro proyecto se han implementado las siguientes clases:

- `contactList.java`: esta clase es la principal de la aplicación. Algunas de sus principales funcionalidades son las siguientes:
    - Cambiar fondo de pantalla con una imagen almacenada en la carpeta `/drawable`:

```
LinearLayout l = (LinearLayout) findViewById(R.id.contactListLayout);
l.setBackgroundResource(R.drawable.wallpaper1);
```

- Cambiar el fondo de pantalla con una imagen de la galería de fotos del teléfono. Tras extraer de las preferencias la dirección donde se almacena esa imagen, se extrae el path de la imagen:

```
Uri uri = Uri.parse(content);
String [] proj={MediaStore.Images.Media.DATA};
Cursor c = getContentResolver().query(uri, proj, null, null, null);
c.moveToFirst();
int index = c.getColumnIndex(MediaStore.Images.Media.DATA);
String path = c.getString(index);
```

  Se crea un nuevo dato de tipo `BitmapDrawable`, que contiene el path de la imagen:

```
Resources res = getResources();
Bitmap thumbnail = (BitmapFactory.decodeFile(path));
BitmapDrawable dr = new BitmapDrawable(res, thumbnail);
```

  Dependiendo de la versión API, se utiliza uno de los siguientes métodos para actualizar el fondo del diseño:

```
if (Build.VERSION.SDK_INT >= 16) {
     l.setBackground(dr.getCurrent());
} else {
     l.setBackgroundDrawable(dr.getCurrent());
}
```

- Sobreescritura del método `onResume()`:

```
super.onResume();
arr = new ArrayList<contactStruct>();
typeOrderPreference();
volumenPreference();
setContentView(R.layout.contact_list);
displaySharedPreferences();
background();
```

- También extrae los números más utilizados del teléfono y crea la interfaz dinámicamente en la pantalla principal en función del número de teléfonos que el usuario desea ver. Estas funcionalidades serán explicadas en los puntos siguientes.

- `orderPersonalized.java`: esta clase permite al usuario seleccionar uno a uno los contactos de su agenda para crear una lista de teléfonos favoritos de un tamaño máximo de 8.
- `adjust.java`: esta actividad se utiliza conjuntamente con la clase `preferenceAdjust.java`, que utiliza fragmentos, para implementar la lista con las preferencias del usuario.

- `gallery.java`: esta última clase crea la galería interna de la aplicación, añadiendo el escuchador `setOnItemClickListener` sobre los elementos de la galería para poder utilizar la imagen seleccionada como fondo de la aplicación. En este método se guarda en las preferencias el número de la imagen seleccionada de la galería interna:

```
gridView.setOnItemClickListener(new OnItemClickListener() {
public void onItemClick(AdapterView<?> parent, View v, int position, long id) {
        SharedPreferences sharedPreferences =
                PreferenceManager.getDefaultSharedPreferences(getApplication());
        SharedPreferences.Editor editor = sharedPreferences.edit();
        editor.putString("imgwhere", "1");
        editor.putString("image", String.valueOf(position));
        editor.commit();
        finish();
        }
});
```

5.3. Proveedores de contenidos

Los proveedores de contenidos ofrecen la posibilidad de leer o escribir datos de otras aplicaciones que de otra manera serían inaccesibles. Se han utilizado los proveedores de contenido para los siguientes casos:

- Obtener la lista de llamadas del teléfono:

```
private void listAllCalls(){
  Uri allCalls = Uri.parse("content://call_log/calls");
  Cursor c = getContentResolver().query(allCalls, null, null, null, null);
  int flag = 0;
  if ((c != null) && c.moveToFirst()){
        do{
        int type =
        Integer.parseInt(c.getString(c.getColumnIndex(CallLog.Calls.TYPE)));
        //1->incoming; 2->out; 3->missed
        ...
        }while(c.moveToNext());
   }
  }
```

- Obtener una imagen del teléfono:

```
Intent intent = new Intent(Intent.ACTION_PICK,
                Uri.parse("content://media/internal/images/media"));
startActivityForResult(intent, CONTACT_PICKER_RESULT);
```

Se almacena en la preferencia lo que se obtiene en `OnActivityResult` y se extrae en la variable `String content`:

```
Uri uri = Uri.parse(content);
String [] proj={MediaStore.Images.Media.DATA};
Cursor c = getContentResolver().query(uri, proj, null, null, null);
c.moveToFirst();
int index = c.getColumnIndex(MediaStore.Images.Media.DATA);
String path = c.getString(index);
```

- Obtener los datos de un contacto tras seleccionarlo de la agenda:

```
public void onActivityResult(int requestCode, int resultCode, Intent data){
String contactId;
Uri contactData = data.getData();
String [] pro = new String[]
                      {ContactsContract.Contacts._ID,
                      ContactsContract.Contacts.DISPLAY_NAME,
                      ContactsContract.Contacts.HAS_PHONE_NUMBER,
                      ContactsContract.Contacts.PHOTO_ID
                      };
```

Obtenemos los datos del usuario concreto:

```
c = getContentResolver().query(contactData, pro, null, null, null);
if (c.moveToFirst()){
if((c.getString(c.getColumnIndex
(ContactsContract.Contacts.HAS_PHONE_NUMBER))).equals("1"){
```

Extraemos el nombre del contacto:

```
name=c.getString(c.getColumnIndex(ContactsContract.Contacts.DISPLAY_NAME));
contactId = c.getString(c.getColumnIndex(ContactsContract.Contacts._ID));
String[] selectionArgs = new String[] {contactId};
```

Para extraer el teléfono, realizamos una segunda consulta sobre el contacto, ya que el teléfono se encuentra en una tabla distinta a la del nombre:

```
c1 =  getContentResolver().query(
               ContactsContract.CommonDataKinds.Phone.CONTENT_URI, null,
               ContactsContract.CommonDataKinds.Phone.CONTACT_ID +" = ?",
               selectionArgs, null);
if(c1.moveToFirst()){
       telef = c1.getString(c1.getColumnIndex(Phone.NUMBER));
   }
  }
}
```

5.4. Ficheros de diseño

Los ficheros de diseño especifican mediante código XML el aspecto de la pantalla de cada actividad. A continuación se describe el fichero de diseño para cada una de las actividades de la sección anterior:

- `contact_list.xml`: corresponde a la actividad `contactList.java`. Este fichero contiene un `RelativeLayout` que se utiliza para colocar los teléfonos de la lista en la pantalla.
- `gallery.xml`: este fichero de diseño corresponde a la actividad `gallery.java` y contiene un `GridView` para poder implementar una galería con dos columnas de separación de dos píxeles.
- `order_personalized.xml`: este fichero de diseño corresponde a la actividad `orderPersonalized.java` y consta de un `LinearLayout` que posee los siguientes elementos:
    - `RelativeLayout`: se utiliza para ir insertando los contactos que el usuario ha seleccionado de su agenda en forma de botones grises.
    - Botones: existen dos botones: uno sirve para añadir un nuevo contacto a la lista personalizada y otro para crear un contacto nuevo en la agenda telefónica.
    - `LinearLayout`: en la parte inferior con los botones de aceptar (guardar los cambios) y cancelar (volver atrás sin realizar ningún cambio).
- `adjust.xml`: este fichero corresponde a la actividad `adjust.java` y consta de un `LinearLayout` para mostrar las preferencias.

5.5. Intenciones

Para la implementación de llamadas a los componentes ya diseñados de Android, como por ejemplo a la agenda de contactos o a la galería de imágenes del teléfono, se han utilizado las intenciones del API de Android. A continuación se muestran los que se han utilizado, y se detallan las funciones que realiza cada uno de ellos.

- Llamada a un número de teléfono: cada uno de los botones de la interfaz tiene un número de teléfono asignado y al pulsar sobre uno de los botones, la variable `tel` del código siguiente toma ese valor. Se crea una nueva intención de tipo `ACTION_CALL`, con el número de teléfono almacenado en la variable `tel` y se realiza la llamada:

```
Intent callIntent = new Intent(Intent.ACTION_CALL);
```

```
callIntent.setData(Uri.parse("tel:"+tel));
startActivity(callIntent);
```

- Insertar un contacto nuevo en la agenda: como se puede observar en el siguiente código, al inicializar una actividad con la acción de `ACTION_INSERT`, se invoca el API de Android, que inserta un nuevo número de teléfono en la agenda:

```
Intent intent = new Intent(Intent.ACTION_INSERT,
    ContactsContract.Contacts.CONTENT_URI);
startActivity(intent);
```

- Crear una actividad de una clase del proyecto: se crea una nueva actividad con una clase ya definida en el proyecto. En el siguiente código se puede observar que con el método `startActivityForResult` se recogen los datos obtenidos al seleccionar uno de los elementos de la nueva actividad:

```
Intent intent = new Intent(getActivity(), gallery.class);
startActivityForResult(intent, CONTACT_PICKER_RESULT);
```

- Mostrar la galería interna del teléfono: en el siguiente código se observa que se puede mostrar la galería de imágenes y seleccionar un elemento cualquiera que se recoge en el método `startActivityForResult`. Para poder entrar en la galería se debe acceder mediante la Uri de la galería:

```
Intent intent = new Intent(Intent.ACTION_PICK,
    Uri.parse("content://media/internal/images/media"));
startActivityForResult(intent, CONTACT_PICKER_RESULT);
```

- Seleccionar un contacto de la lista del teléfono: para poder seleccionar un contacto del teléfono basta con utilizar `ContactsContract.Contacts` en el `Intent`. Anteriormente se utilizaba simplemente `Contacts`, pero desde la versión de Android 2.0 (API 5) se quedó obsoleto:

```
Intent addContact = newIntent(Intent.ACTION_PICK,
    ContactsContract.Contacts.CONTENT_URI);
startActivityForResult(addContact, CONTACT_PICKER_RESULT);
```

5.6. Fragmentos

La aplicación utiliza fragmentos, ya que estos permiten una mayor flexibilidad a la hora de realizar la interfaz dependiendo del tamaño de pantalla del dispositivo. Los fragmentos se han implementado en el uso de las preferencias.

5.6.1. Preferencias

Las preferencias están implementadas en la clase `preferenceAdjust.java`, que en lugar de extender de `preferenceActivity`, extiende de `preferenceFragment`.

Para utilizar las preferencias se utiliza una actividad intermedia que en este caso es `adjust.java`, la cual utiliza los fragmentos y transacciones:

```
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.adjust);
        FragmentManager fragmentManager = getFragmentManager();
        FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();
        preferenceAdjust fragment = new preferenceAdjust();
        fragmentTransaction.replace(android.R.id.content, fragment);
        fragmentTransaction.commit();
}
```

También define las constantes de las preferencias con un valor inicial por defecto:

```
static String NUMBER_KEY = "number";
static String NUMBER_DEFAULT = "4";
```

Cada una de las preferencias está declarada en el fichero XML `adjust.xml`, almacenado en la carpeta `res/xml`:

```
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">

      <PreferenceCategory
      android:title="@string/general">
               <ListPreference
               android:key="number"
               android:summary="@string/numberSummary"
               android:title="@string/numberTitle"
               android:entries="@array/listNum"
               android:entryValues="@array/listNum_values"/>
               <Preference
               android:key="order"
               android:summary="@string/orderSummary"
               android:title="@string/orderTitle"
               android:entries="@array/listOrder"
               android:entryValues="@array/listOrder_values"/>
               <Preference
               android:key="image"
               android:summary="@string/imageSummary"
               android:title="@string/imagePrefTitle">
               </Preference>
      </PreferenceCategory>

      <PreferenceCategory
      android:title="@string/audioTitle">
               <ListPreference
```

```
            android:key="volumen"
            android:summary="@string/volumeSummary"
            android:title="@string/volumeTitle"
            android:entries="@array/listVol"
            android:entryValues="@array/listVol_values"/>

            <RingtonePreference
            android:key="ringtone"
            android:summary="@string/ringtoneSummary"
            android:title="@string/ringtoneTitle"/>
    </PreferenceCategory>
</PreferenceScreen>
```

Como se puede ver en el fichero XML, existen varios tipos de preferencias, que son los siguientes:

- `ListPreference`: a partir de una lista de valores almacenada en el fichero `array.xml`, se puede seleccionar uno de esos valores. A continuación se muestra un fragmento de este fichero:

```
<string-array name="listOrder">
        <item>Realizadas</item>
        <item>Recibidas</item>
        <item>Totales</item>
        <item>Personalizado</item>
</string-array>
<string-array name="listOrder_values">
        <item>1</item>
        <item>2</item>
        <item>3</item>
        <item>4</item>
</string-array>
```

- `Preference`: es la preferencia genérica que permite modificar el contenido libremente (poner listas, un solo valor, un texto, etc.).
- `RingtonePreference`: preferencia para el tono del móvil.

Para almacenar las preferencias por grupos dentro de este fichero XML, se utiliza la etiqueta `PreferenceCategory`.

En la clase `preferenceadjust.java` se implementa el escuchador de eventos `onPreferenceClick`, y para ello, se añade el evento `setOnPreferenceClickListener` a las preferencias `order` e `image`, las cuales deben interactuar con el usuario.

En el método `onCreate` se asigna el escuchador a dos de las preferencias:

```
public void onCreate (Bundle savedInstanceState){
        super.onCreate(savedInstanceState);
        addPreferencesFromResource(R.xml.adjust);
        mypref = findPreference("image");
        mypref.setOnPreferenceClickListener(this);
        person = findPreference("order");
        person.setOnPreferenceClickListener(this);
}

@Override
public boolean onPreferenceClick(Preference preference){
```

En esta preferencia se crea un diálogo para que el usuario pueda elegir entre las dos posibilidades para elegir el fondo de pantalla:

```
      if(preference.getKey().equals("image")){
        AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
        builder.setTitle(R.string.imageTitle)
                .setItems(R.array.listImage, new DialogInterface.OnClickListener() {
                        public void onClick(DialogInterface dialog, int item) {
                         if(item == 0){
                                Intent intent = new Intent(getActivity(), gallery.class);
                                startActivityForResult(intent, CONTACT_PICKER_RESULT);
                         }else{
                                Intent intent = new Intent(Intent.ACTION_PICK,
                                Uri.parse("content://media/internal/images/media"));
                                startActivityForResult(intent, CONTACT_PICKER_RESULT);
                         }
                        }
                });
                builder.show();
        }
```

En esta preferencia, en lugar de elegir una opción, se redirige a la pantalla de `orderPersonalized.java`:

```
if(preference.getKey().equals("order")){
      ...
      public void onClick(DialogInterface dialog, int item) {
                SharedPreferences.Editor editor = sharedPreferences.edit();
                editor.putString("order", String.valueOf(item+1));
                editor.commit();
                if(item == 3){
                        startActivity(new Intent(getActivity(), orderPersonalized.class));
                }
                dialog.dismiss();
      }
}
```

Finalmente, las preferencias pueden ser leídas y modificadas desde cualquier otra actividad de la aplicación.

Para leer la preferencia `image`, que contiene el número de la imagen de la galería de la aplicación:

```
SharedPreferences sharedPreferences =
      PreferenceManager.getDefaultSharedPreferences(this);
String content = sharedPreferences.getString(adjust.IMAGE_KEY, adjust.IMAGE_DEFAULT);
```

Para modificar la preferencia `image` con el valor de la posición donde se encuentra:

```
SharedPreferences sharedPreferences =
      PreferenceManager.getDefaultSharedPreferences(getApplication());
SharedPreferences.Editor editor = sharedPreferences.edit();
editor.putString("image", String.valueOf(position));
editor.commit();
```

5.7. Diálogos

Para notificar al usuario alertas o darle a elegir una opción para poder continuar, se han implementado diálogos. Un ejemplo de ello se produce cuando el usuario desea cambiar de fondo y tiene que elegir entre la galería del teléfono o las imágenes de la aplicación.

Éste es un diálogo de ayuda, que no requiere interacción del usuario:

```
AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setTitle(R.string.about);
builder.setMessage(R.string.aboutContent);
builder.show();
```

Este diálogo, tras seleccionar el orden personalizado, inicia una nueva actividad (`orderPersonalized.java`):

```
AlertDialog.Builder builder = newAlertDialog.Builder(getActivity());
builder.setTitle(R.string.orderTitle).setSingleChoiceItems
       (R.array.listOrder, orderType-1, newDialogInterface.OnClickListener() {
       public void onClick(DialogInterface dialog, int item) {
                     SharedPreferences.Editor editor = sharedPreferences.edit();
                     editor.putString("order", String.valueOf(item+1));
                     editor.commit();
                     if(item == 3){
                            startActivity(new Intent(getActivity(),
                                   orderPersonalized.class));
                     }
                     dialog.dismiss();
       }
```

```
});
builder.show();
```

5.8. Menú

El menú de la aplicación está diseñado para poder ser utilizado tanto de la barra de acción como del botón de menú (en dispositivos más antiguos).

El menú está almacenado en la carpeta `res/menu`. En esta carpeta se crea el fichero XML correspondiente. En este caso se llama `main_menu.xml` y su estructura es la siguiente:

```
<?xmlversion="1.0"encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">

      <item
      android:title="@string/ajustes"
      android:id="@+id/ajustes"/>
      <item
      android:title="@string/addNumber"
      android:id="@+id/menu_addCont"/>
      <item
      android:title="@string/about"
      android:id="@+id/about"/>
</menu>
```

El menú se declara en la clase `contactList.java` y se definen las acciones que han de realizar cada uno de sus elementos:

```
public boolean onCreateOptionsMenu(Menu menu){
      super.onCreateOptionsMenu(menu);
      MenuInflater inflater = getMenuInflater();
      inflater.inflate(R.menu.main_menu, menu);
      return true;}

public boolean onOptionsItemSelected(MenuItem item){

  switch  (item.getItemId()){
      case R.id.ajustes:
        startActivity(new Intent(this, adjust.class));
        return true;
      case R.id.menu_addCont:
        Intent intent = new Intent(Intent.ACTION_INSERT,
                        ContactsContract.Contacts.CONTENT_URI);
        startActivity(intent);
                return true;
        case R.id.about:
                AlertDialog.Builder builder = new AlertDialog.Builder(this);
                builder.setTitle(R.string.about);
                builder.setMessage(R.string.aboutContent);
                builder.show();
                return true;
        }
        return super.onOptionsItemSelected(item);
      }
```

5.9. Internacionalización

Para que la aplicación sea accesible a un mayor número de usuarios, está traducida al inglés. Para ello, se han almacenado todas las cadenas de texto en el fichero `strings.xml`, que se localiza en la carpeta `res/values`. En este fichero cada cadena de texto tiene un identificador, que es el que se utiliza en lugar de dicha cadena de texto.

Cada una de las cadenas de texto están declaradas de la siguiente manera:

```
<string name="create_contact">Crear contacto nuevo</string>
```

Si se quiere utilizar una cadena de texto desde un fichero XML, la manera de utilizarla sería:

```
android:text="@string/create_contact".
```

Si en lugar de un fichero XML se quiere utilizar una cadena de texto almacenada en `strings.xml` en un fichero `.java`, debe hacerse de esta forma:

```
.setTitle(R.string.create_contact);
```

Para realizar la traducción, se crea la carpeta `res/values-XX`, donde `XX` es el prefijo del idioma a traducir; en este caso, se utiliza "en" (inglés). Dentro de esta carpeta se crea un fichero similar al original (`strings.xml`) y, utilizando los mismos identificadores, se traducen las cadenas de texto.

Finalmente, se ha realizado también la traducción de las listas de elementos, como es la lista de posibles llamadas. Estas listas están almacenadas en un fichero XML llamado `array.xml`, que está en la misma carpeta que `strings.xml`. El mecanismo a seguir para la traducción es el mismo que con el otro fichero.

5.10. Interfaz

La interfaz se ha implementado siguiendo el modelo de diseño, y para ello se han diseñado varios componentes en ficheros de diseño XML dentro de la carpeta `drawable`.

`button.xml`: se define un botón con el fondo blanco definido en la etiqueta `gradient` con el formato #Alfa-Red-Green-Blue de la siguiente manera: `#AARRGGBB`.

Para redondear las esquinas se utiliza la etiqueta `corners`, que con la propiedad `android:radius` se redondean todas las esquinas.

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
```

```
        android:shape="rectangle">

        <gradient
                android:startColor="#FFFFFFFF"
                android:endColor="#FFFFFFFF"
                android:angle="45"/>
        <padding
                android:left="7dp"
                android:top="7dp"
                android:right="7dp"
                android:bottom="7dp" />
        <corners
                android:radius="30dp" />
</shape>
```

- `button_left.xml`: se define un botón similar al anterior, pero solo se redondean las esquinas del lado derecho. El lado izquierdo se expande hasta el final de la pantalla.

```
<corners
        android:topRightRadius="30dp"
        android:bottomRightRadius="30dp" />
```

- `button_right.xml`: se define un botón similar al del fichero button.xml pero solo se redondean las esquinas del lado izquierdo. El lado derecho se expande hasta el final de la pantalla.

```
<corners
        android:topLeftRadius="30dp"
        android:bottomLeftRadius="30dp"/>
```

- `rectangle.xml`: se define un botón similar a los anteriores pero de color gris y con las esquinas menos redondeadas que los anteriores: 8dp en lugar de 30dp.

```
<gradient
        android:startColor="#FFCFCFCF"
        android:endColor="#FFCFCFCF"
        android:angle="45"/>
<corners android:radius="8dp" />
```

- Galería: Para el diseño del fondo de la aplicación, el usuario puede elegir entre las dos posibilidades de fondos de la galería del teléfono o fondos predefinidos de la aplicación. Las imágenes creadas para estos fondos están almacenadas en la carpeta `/drawable` con formato `.jpg`.

  La galería para poder mostrar estas imágenes y que el usuario seleccione una está creada en la clase `imageAdapter.java`, la cual crea la galería de cuatro imágenes teniendo en cuenta el tamaño de la pantalla. Esta clase es llamada por

`gallery.java`, que es la que la crea junto al evento de selección de una de las imágenes para almacenarla en las preferencias.

```
public class imageAdapter extends BaseAdapter{

	private Context mContext;
	public imageAdapter(Context c){
		mContext = c;
	}
	@Override
	publicint getCount() {
		// TODO Auto-generated method stub
		return mThumbIds.length;
	}
	@Override
	public Object getItem(int position) {
		// TODO Auto-generated method stub
		return null;
	}
	@Override
	public long getItemId(int position) {
		// TODO Auto-generated method stub
		return 0;
	}
	@Override
	public View getView(int position, View convertView, ViewGroup parent) {
		ImageView imageView;
		if (convertView == null) {
			imageView = new ImageView(mContext);
```

Extraemos la altura y anchura de la pantalla:

```
int widthPix = (int) mContext.getResources().getDisplayMetrics().widthPixels;
int heightPix = (int) mContext.getResources().getDisplayMetrics().heightPixels;
int width = widthPix/2;
int height = heightPix/2;
```

Damos la altura y anchura al contenedor:

```
imageView.setLayoutParams(new GridView.LayoutParams(width, height));
 } else {
imageView = (ImageView) convertView;
 }
imageView.setImageResource(mThumbIds[position]);
return imageView;
}
```

Se listan las imágenes que queremos que se vean en la galería:

```
private Integer[] mThumbIds = {
R.drawable.wallpaper1, R.drawable.wallpaper2, R.drawable.wallpaper3,
	R.drawable.wallpaper4
	};
}
```

- Creación de la interfaz de la pantalla principal: en el método contactList() se crea la interfaz dinámicamente en función del número de contactos que ha seleccionado el usuario o si ha creado una lista personalizada.

  Se crea un array de botones del tamaño de la lista y se asignan los valores del array de números de teléfono a cada botón:

```
Button[] buttons = new Button[num];
for(k=0; k<num && k<index; k++){
        nombre = arr.get(index-k-1).getNom();
        telef = arr.get(index-k-1).getnumTel();
        buttons[k] = new Button(this);
        buttons[k].setId(k+1);
        if(k == 0){
                RelativeLayout.LayoutParams lp = new
RelativeLayout.LayoutParams(RelativeLayout.LayoutParams.WRAP_CONTENT,
  RelativeLayout.LayoutParams.WRAP_CONTENT);
```

  Si el total de números a mostrar es menor o igual a cuatro, se mostrará, centrados en la pantalla principal:

```
if(num <= 4){
buttons[k].setBackgroundResource(R.drawable.button);
lp.addRule(RelativeLayout.CENTER_HORIZONTAL);
lp.setMargins(10, 10, 10, 10);}
```

  Por el contrario, si son más de cuatro números, se mostrarán cuatro a la izquierda y cuatro a la derecha, por lo que este primer número lo ponemos a la izquierda y centrado en la pantalla:

```
else{
buttons[k].setBackgroundResource(R.drawable.button_left);
lp.addRule(RelativeLayout.ALIGN_PARENT_LEFT);
lp.setMargins(0, 10, 10, 10);
}
lp.addRule(RelativeLayout.CENTER_VERTICAL);
buttons[k].setLayoutParams(lp);
```

  Este código se reproduce para cada uno de los botones, situándolos todos alrededor del botón creado anteriormente, de manera que todos quedan centrados en la pantalla y alineados unos con otros.

5.11. Otras especificaciones y fichero de manifiesto

Para que la aplicación se inicie al arrancar el teléfono se ha creado una clase llamada `BootUpReceiver.java`, la cual crea un *Intent* de la clase principal y le añade un "flag" o propiedad que hace que la actividad principal se convierta en la primera en ser lanzada. El código es el siguiente:

```
package es.uam.eps.proyecto;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;

public class BootUpReceiver extends BroadcastReceiver{
      @Override
      publicvoid onReceive(Context context, Inent intent) {
        Intent i = new Intent(context, contactList.class);
               i.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
               context.startActivity(i);
      }
}
```

Por otra parte, esta clase también es definida en el fichero de manifiesto con la característica de que se define como `receiver`, ya que es un receptor de difusión. La declaración en el fichero `AndroidManifest.xml` se realiza dentro de la etiqueta `application`, y es la siguiente:

```
<receiver android:enabled="true"android:name="es.uam.eps.proyecto.BootUpReceiver"
android:permission="android.permission.RECEIVE_BOOT_COMPLETED">
      <intent-filter>
      <action android:name="android.intent.action.BOOT_COMPLETED" />
      <category android:name="android.intent.category.DEFAULT" />
      </intent-filter>
</receiver>
```

Otro elemento utilizado es el control de volumen del teléfono. Para recoger el volumen actual del teléfono se utiliza el siguiente código:

```
int vol;
AudioManager audio = (AudioManager)getSystemService(Context.AUDIO_SERVICE);
vol = audio.getRingerMode();
```

Donde `vol` puede tomar los siguientes valores: `RINGER_MODE_NORMAL`, `RINGER_MODE_SILENT` o `RINGER_MODE_VIBRATE`, que equivalen a los valores constantes 2, 0 y 1 respectivamente.

Para cambiar el volumen del teléfono en función de lo guardado en las preferencias, se realizan las siguientes acciones:

Primero se extrae el sonido medio del teléfono:

```
AudioManager audio = (AudioManager) getSystemService(Context.AUDIO_SERVICE);
int maxVolume = audio.getStreamMaxVolume(AudioManager.STREAM_RING);
```

A continuación, dependiendo del valor almacenado en las preferencias (`volType`), se aplican los cambios mediante los métodos `setRingerMode` y `setStreamVolume`:

```
if(volType == 1){//alto
      audio.setRingerMode(AudioManager.RINGER_MODE_NORMAL);
      audio.setStreamVolume(AudioManager.STREAM_RING, maxVolume,
                        AudioManager.FLAG_PLAY_SOUND);
      }elseif(volType == 2){//medio
                audio.setRingerMode(AudioManager.RINGER_MODE_NORMAL);
                audio.setStreamVolume(AudioManager.STREAM_RING, maxVolume/2,
                                  AudioManager.FLAG_PLAY_SOUND);
      }else if(volType == 3){//bajo
                audio.setRingerMode(AudioManager.RINGER_MODE_NORMAL);
                audio.setStreamVolume(AudioManager.STREAM_RING, maxVolume/4,
                                  AudioManager.FLAG_PLAY_SOUND);
      }else if(volType == 4){//vibración
                audio.setRingerMode(AudioManager.RINGER_MODE_VIBRATE);
                audio.setStreamVolume(AudioManager.STREAM_RING, 0,
                                  AudioManager.FLAG_VIBRATE);
      }else{//5->silencio
                audio.setRingerMode(AudioManager.RINGER_MODE_SILENT);
      }
}
```

Finalmente, en el fichero de manifiesto se declaran las propiedades de la aplicación. En este se especifica que la versión mínima de API es 14 (correspondiente a Android 4.0). Se incluye la propiedad `portrait` a todas las actividades para evitar que la pantalla pueda rotar.

```
<activity
        android:name="es.uam.eps.proyecto.adjust"
        android:label=""
        android:screenOrientation="portrait">
```

Para poder realizar llamadas, y otras funcionalidades del teléfono, es necesario que el fichero de manifiesto tenga los permisos necesarios para ello. Se han incluido los siguientes permisos en la aplicación:

Permiso para leer los contactos del teléfono:

```
<uses-permission android:name="android.permission.READ_CONTACTS"/>
```

Permiso para realizar llamadas:

```
<uses-permission android:name="android.permission.CALL_PHONE"/>
```

Permiso para leer el estado del teléfono:

```
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
```

Permiso para acceder a la vibración:

```
<uses-permission android:name="android.permission.VIBRATE"/>
```

Permite a la aplicación recibir un receptor de difusión:

```
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
```

Permiso para leer el listado de llamadas:

```
<uses-permission android:name="android.permission.READ_CALL_LOG"/>
```

Permiso para escribir en los contactos o añadir uno nuevo:

```
<uses-permission android:name="android.permission.WRITE_CONTACTS"/>
```

Permiso para escribir en el registro de llamadas del usuario:

```
<uses-permission android:name="android.permission.WRITE_CALL_LOG"/>
```

Permiso para escribir en el almacenamiento externo:

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
```

6 | Pruebas

Para verificar el correcto funcionamiento de la aplicación FastCall se han realizado varios tipos de pruebas. Dichas pruebas son necesarias debido a la amplia gama de teléfonos disponibles en el mercado, y sin las mismas la aplicación podría fallar en cualquier momento o no tener un comportamiento esperado. Las pruebas están divididas en pruebas unitarias, de integración, de sistema; finalmente, se realizaron pruebas de validación con varios usuarios para comprobar el grado de satisfacción de los mismos.

6.1. Pruebas unitarias

Las pruebas unitarias, también llamadas pruebas de módulos o pruebas de clases, son aquellas que verifican cada uno de los módulos de la aplicación. Su objetivo principal es encontrar fallos que son fácilmente subsanables debido a que verifican el funcionamiento de los módulos por separado. A continuación, se muestran las pruebas más representativas del proyecto FastCall.

- Prueba 01: listado de números telefónicos más llamados.

  Para extraer el listado de llamadas del teléfono, hay que analizar el método `listAllCalls()` (dentro de la clase `contactList.java`), el cual extrae cada una de las llamadas y verifica si son realizadas, recibidas o perdidas. El resultado esperado es un array del tipo de llamadas deseadas.

  Pruebas realizadas:

  - En caso de existir varias llamadas a varios números, se extrae el orden correctamente.
  - Si el teléfono no ha realizado ninguna llamada, se muestra la lista de números vacía.

- Si se han llamado a números con prefijos o extranjeros, funciona exactamente igual que con números nacionales o sin prefijo.

- Prueba 02: extracción de datos de un contacto seleccionado de la agenda.

  En la clase `orderPersonalized.java` se implementa el código para obtener los datos de los contactos de la agenda. Para verificar que se extraen los datos correctamente, se analiza el método `onActivityResult`.

  Pruebas realizadas:

  - Si se selecciona un contacto con un número de teléfono, se extraen el nombre y el número de teléfono.
  - Si se selecciona un contacto sin teléfono asociado, no se añade dicho contacto y se notifica al usuario que el contacto no tiene un número de teléfono.
  - Si el contacto no tiene un nombre asociado, se mostrará el número de teléfono.

- Prueba 03: aplicar un fondo de pantalla.

  El método `background()`, dentro de la clase `contactList.java`, extrae de las preferencias un tipo de fondo de pantalla (de la aplicación o de la galería). Esto es necesario para aplicar el nuevo fondo ya que, si es de la aplicación, hay que extraer cuál de los cuatro predefinidos es, y, en caso contrario, se deberá extraer la URI de la imagen (almacenado en las preferencias).

  Pruebas realizadas:

  - Si se selecciona un fondo de la aplicación, funciona correctamente para todos los casos.
  - Si se selecciona un fondo de la galería en un dispositivo con API inferior a 16, se utiliza un método que en APIs superiores es reemplazado por otro.
  - En caso de arrancar la aplicación por primera vez o que los datos no estén guardados correctamente en las preferencias, existe un fondo predefinido.

- Prueba 04: mostrar la lista de teléfonos más llamados o lista personalizable.

El método `displaySharedPreferences()` extrae cada uno de los elementos de la lista donde están guardados los contactos más llamados o preferidos, y en función del número a mostrar, indica en pantalla los deseados.

Pruebas realizadas:

- Si se quieren mostrar el mismo número de contactos que existentes en la lista.
- Si se quieren mostrar menos contactos que los existentes en la lista.
- Si se quieren mostrar más contactos de los existentes en la lista.
- Si la lista está vacía.

6.2. Pruebas de integración

Las pruebas de integración son aquellas que verifican que un conjunto de partes del software funcionan correctamente. Estas pruebas combinan los módulos probados en las pruebas unitarias; por ello, se deben realizar una vez aprobadas las pruebas anteriores. A continuación se explicarán las pruebas más representativas del proyecto FastCall.

- Prueba 01: guardado de preferencias.

  Las preferencias se aplican en él al salir de la actividad donde están contenidas, por lo que es necesario probarlas junto a otras clases.

  La actividad `contactList.java` actualiza el volumen, fondo de pantalla, orden y número del listado. Por ello se prueban combinaciones entre varias clases.

  Pruebas realizadas con las clases `contactList.java` y adjust.java:

  - El volumen, orden, número de contactos y la nueva imagen de fondo se cambian correctamente al volver a la clase `contactList.java`.
  - Si se cambian varias preferencias antes de volver a la clase `contactList.java`, se actualizan todos los cambios.
- Prueba 02: guardado y extracción de la lista personalizada.

  Una vez que el usuario tiene seleccionados los contactos que desea en la lista personalizada y guarda los cambios, estos se guardan en una preferencia de tipo cadena de texto, que en realidad es un JSON. El guardado se realiza en el

evento del botón de aceptar de la clase `orderPersonalized.java` y la extracción en el método `listPersonalized()` de la clase `contactList.java`.

Pruebas realizadas:

- Si se inserta un contacto, se relacionan correctamente nombre y teléfono.
- Si se inserta más de un contacto, la relación nombre y teléfono es correcta.
- Si no se inserta ningún contacto, se guarda una cadena de texto vacía y no se muestra ningún contacto.

- Prueba 03: obtención de una de las imágenes predefinidas de la aplicación.

Para la realización de esta prueba se combinan las clases `gallery.java` e `imageAdapter.java`. Se verifica que el usuario, al seleccionar una de las cuatro imágenes predefinidas de la galería, lo ha hecho correctamente.

Pruebas realizadas:

- Si el usuario selecciona una de las imágenes, se guarda correctamente en las preferencias.
- Si no se selecciona ninguna imagen, se mantiene la que está guardada en las preferencias.

6.3. Pruebas de sistema

Las pruebas del sistema verifican el correcto funcionamiento del sistema software. Estas pruebas comprueban que se satisfacen cada uno de los requisitos funcionales de la aplicación. A continuación se explicarán varias de las pruebas realizadas para la aplicación FastCall.

- Prueba 01: funcionamiento estándar en Android 4.0.4.

  Pruebas realizadas:

  - Se ha probado la aplicación con esta versión en una pantalla pequeña (Sony Tipo).

- Prueba 02: funcionamiento estándar en Android 4.1.

  Pruebas realizadas:

  - Se ha probado la aplicación en Android 4.1.2 con un Nexus 4.

- Prueba 03: funcionamiento estándar en Android 4.2.

  Pruebas realizadas:

  - Se ha probado en la máquina virtual de Eclipse con 3.4” y versión 4.2.2.

- Prueba 04: funcionamiento estándar en Android 4.3.

  Pruebas realizadas:

  - Se ha probado en la máquina virtual de Eclipse con 5.1" y versión 4.3.
- Prueba 05: funcionamiento estándar en Android 4.4.

  Pruebas realizadas:

  - Se ha probado en Nexus 5 con versión 4.4.2.

6.4. Pruebas de aceptación

Las pruebas de aceptación son aquellas que se realizan por clientes que determinan la aceptación o rechazo del producto desarrollado. Estas pruebas se realizan antes de que la aplicación sea instalada en un entorno de producción. Para verificar todo lo anterior, se han realizado varias pruebas con distintos usuarios para evaluar su grado de satisfacción.

Encuesta 01: mujer, 77 años.

P-¿Sabe Ud. manejar un teléfono táctil?

R-No tengo ni idea, solo utilizo este móvil (móvil antiguo de pantalla pequeña y teclas) para llamar. Solo sé darle a este botón y llamar (acceso a la agenda directo).

P-¿Sabe bloquearlo?

R-No, tuve que llamar a mi hijo por el teléfono fijo porque no sabía abrirlo.

P-¿Qué le parece si Ud. utiliza este móvil de última generación como uno nuevo? (se le muestra el funcionamiento y se le indica que solo ha de pulsar un botón para llamar)

R-Me parece muy bien, pero no se configurar eso.

P-No se preocupe, eso se lo configuraría su hijo o nieta. Una última pregunta, ¿utilizaría este móvil de última generación con esta aplicación de solo pulsar para llamar?

R-Si claro, es muy fácil de usar así.

P-Muchas gracias por su tiempo.

Encuesta 02: mujer, 75 años.

P- ¿Sabe Ud. manejar un teléfono táctil?

R- No, yo solo manejo este (móvil similar al de la encuesta anterior) y solo para llamar.

P- ¿Sabe bloquearlo?

R- Sí.

P-¿Qué le parece si Ud. utiliza este móvil de última generación como uno nuevo? (se le muestra el funcionamiento y se le indica que solo ha de pulsar un botón para llamar).

R- Me parece algo muy útil, pero yo no quiero manejar este tipo de aparatos, no me gusta el teléfono móvil, este lo tengo porque me lo regaló mi hija y no me gusta usarlo.

6.5. Resultados

Al realizar las pruebas unitarias se han ido arreglando los errores a medida que iban surgiendo, siendo así mucho más fácil de detectar los errores en la lógica del programa.

Las pruebas de integración han servido para verificar que las preferencias eran consistentes y la transición de una pantalla a otra coherente.

Finalmente, las pruebas de sistema han verificado que, a medida que iban evolucionando las versiones de Android, se han añadido algunos nuevos permisos de Android en el fichero de manifiesto, por lo que han sido muy beneficiosas.

Una vez hechas todas las pruebas, se concluye que tanto las pruebas unitarias como de integración y de sistema son satisfactorias para todos los casos, no produciéndose fallos o errores en ninguno de ellos.

Las pruebas de aceptación han sido realmente útiles, ya que se comprueba que los usuarios finales serían capaces de utilizar estos nuevos dispositivos con la aplicación desarrollada.

7 Conclusiones

Como se ha podido leer en este informe, Android es un sistema operativo multiplataforma, que actualmente es el más utilizado en el mercado y cuenta con múltiples versiones y dispositivos que lo ejecutan.

Este proyecto está programado para funcionar en Android 4.0 y versiones superiores. Incluso no incluyendo las versiones anteriores a Android 4.0, se obtiene una cuota de mercado del 77,4%, cuota que va en aumento si tenemos en cuenta que cada vez se venden más dispositivos con las últimas versiones, o se actualizan dispositivos de versiones anteriores a estas nuevas.

El proyecto ha sido diseñado pensando en personas mayores o con dificultades para manejar teléfonos móviles inteligentes que, si bien son muy potentes, suponen un desafío infranqueable para estos colectivos de personas.

FastCall presenta una interfaz gráfica extremadamente sencilla que convierte la tarea de realizar una llamada en una experiencia muy amigable que elimina la frustración que el uso de los teléfonos inteligentes puede ocasionar en ciertos colectivos de personas.

8 Trabajo futuro

Una de las consideraciones a tener en cuenta a la hora de programar en Android es que el sistema operativo se actualiza con mucha frecuencia, bien con una nueva versión o bien actualizando una existente para mejorar su compatibilidad con algunas aplicaciones o dispositivos.

Por ello, es importante que cada vez que se actualiza una nueva versión se debe estudiar el API de Android, para ver qué métodos han sido actualizados, incorporados o cuáles se han quedado obsoletos. Si no se realiza esta tarea de mantenimiento podría haber problemas de compatibilidad con los terminales, o que la aplicación deje de funcionar tal y como se espera, ya que un método obsoleto en un API determinado puede no funcionar en un API superior.

Además de esta meticulosa tarea de mantenimiento, sería interesante incorporar imágenes a FastCall para identificar a los contactos. De esta forma, en vez de identificar a los contactos por su nombre, se podría simplemente pulsar sobre su fotografía o sobre la imagen que tuviera asignada en la lista de contactos.

Otro posible trabajo futuro sería la incorporación de nuevos idiomas en la aplicación. Actualmente está disponible en español y en inglés, pero podrían añadirse otros idiomas como el francés o el italiano sin dificultad, gracias a que todas las cadenas de texto han sido programadas como recursos para favorecer la internacionalización de la aplicación.

Referencias

[1] http://www.xatakandroid.com/mercado/android-eleva-su-cuota-de-mercado-hasta-el-80[visitado el 15 de diciembre de 2013]

[2] http://androideity.com/2011/07/04/arquitectura-de-android/[visitado el 2 de diciembre de 2013]

[3] https://sites.google.com/site/swcuc3m/home/android/generalidades/2-2-arquitectura-de-android[visitado el 2 de diciembre de 2013]

[4] http://es.wikipedia.org/wiki/Anexo:Historial_de_versiones_de_Android[visitado el 5 de diciembre de 2013]

[5]http://www.ticbeat.com/sim/la-historia-de-android-infografia/ [visitado el 29 de noviembre de 2013]

[6] http://developer.android.com/about/dashboards/index.html [visitado el 28 de diciembre de 2013]

[7] http://www.somosbinarios.es/versiones-de-android-1a-parte/ [visitado el 6 de diciembre de 2013]

[8] http://www.lanacion.com.ar/1623151-android-cumple-5-anos-y-10-tambien[visitado el 30 de noviembre de 2013]

[9] http://androidayuda.com/2013/12/02/android-sigue-arrasando-en-espana-con-mas-del-90-de-cuota-de-mercado/[visitado el 28 de noviembre de 2013]

[10] http://www.elandroidelibre.com/2013/11/android-es-samsung-o-samsung-es-android-los-datos-de-cuota-de-mercado-diseccionadas.html [visitado el 12 de diciembre 2013]

[11] http://pruebasdelsoftware.wordpress.com/[visitado el 2 de enero de 2014]

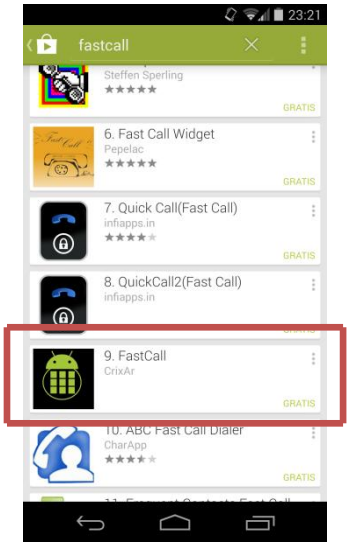
[12] http://www.androidcurso.com/index.php/tutoriales-android/42-unidad-9-almacenamiento-de-datos/311-utilizando-la-clase-contentprovider [visitado el 5 de enero de 2014]

[13] http://www.kantarworldpanel.com/Global/News/Apple-iPhone-5S-outsells-5C-three-to-one-in-Great-Britain [visitado el 1 de diciembre de 2013]

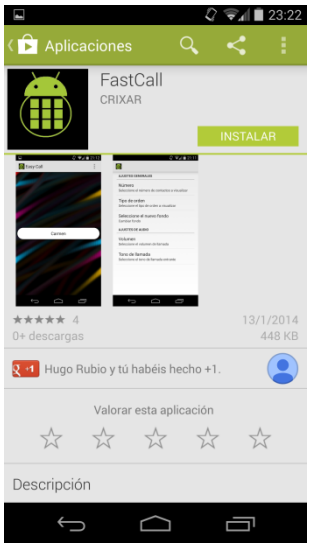
Anexo A

Descarga e instalación de la aplicación

Para poder instalar la aplicación basta con tener una cuenta en Google. Una vez que la tenemos, accedemos a la aplicación Google Play Store, que viene por defecto en todos los móviles Android.



Una vez dentro de la aplicación, realizamos la búsqueda por nombre, ponemos en el buscador FastCall y seleccionamos aplicaciones. Una vez que vemos el icono de la aplicación, la seleccionamos y pulsamos en el botón de descargar.

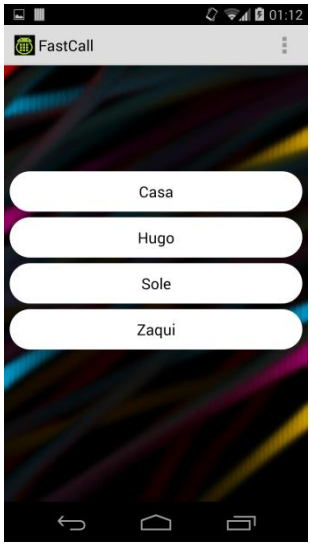


Google Play Store descarga e instala la aplicación automáticamente.

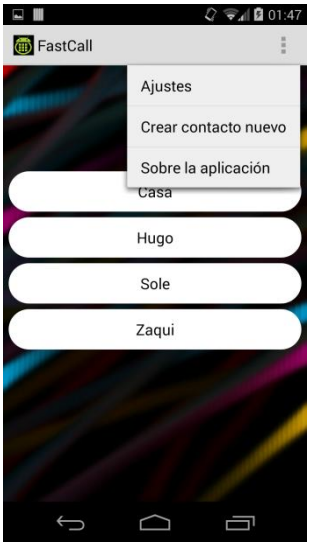
Anexo B

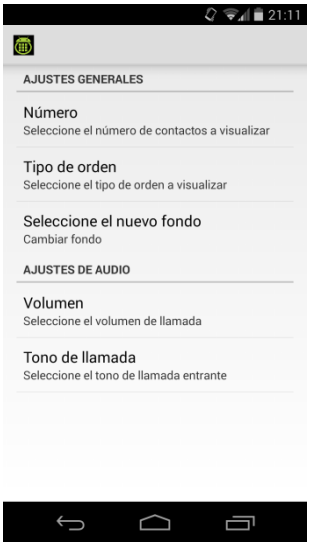
Manual de usuario de la aplicación.

Al arrancar la aplicación por primera vez, se mostrarán los cuatro números de teléfono más utilizados por el usuario. Basta pulsar uno de ellos para efectuar una llamada telefónica:

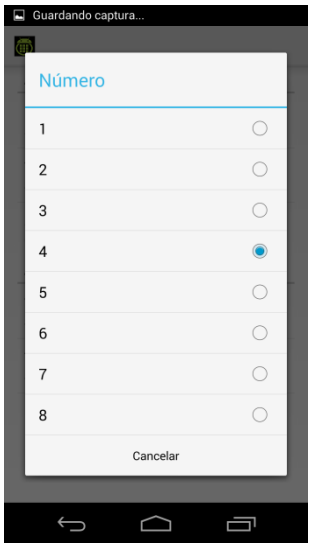


Para cambiar los ajustes pulsamos sobre el botón de menú y seleccionamos ajustes.

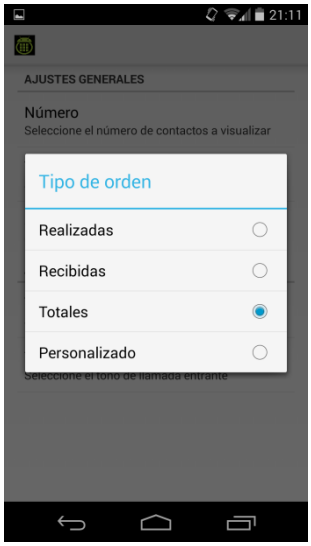




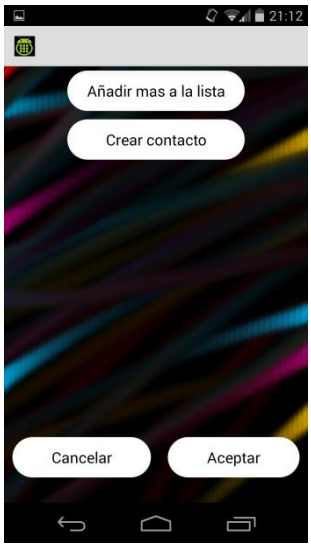
El primer ajuste es número. Este valor muestra el número de contactos que se verán en la pantalla principal.



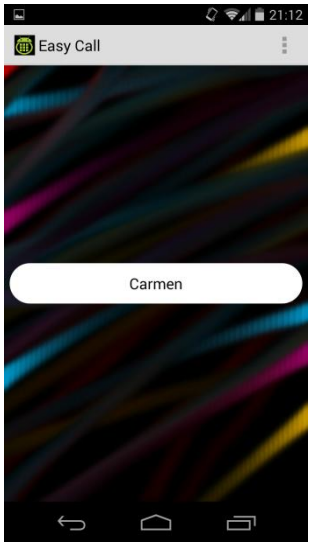
El segundo ajuste es tipo de orden. Este ajuste nos permite elegir el orden en el que queremos que se muestren los contactos de la pantalla principal.



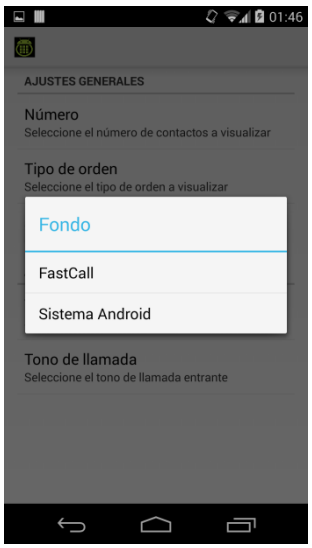
La opción por defecto es Totales, pero si se selecciona la opción Personalizado, se accederá a la siguiente pantalla, en la cual podemos crear un contacto nuevo en la agenda o ir creando nuestra lista de teléfonos personalizada. Se pueden añadir hasta ocho números distintos a la lista:

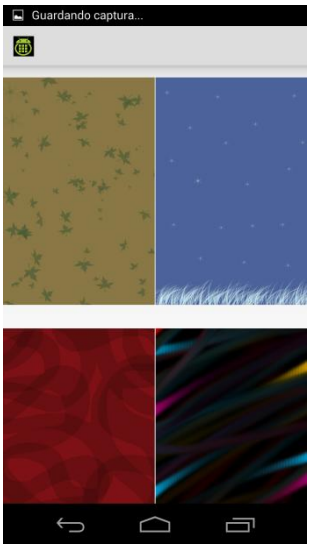


Cuando aceptamos o cancelamos, volvemos a las preferencias, y podemos volver a la pantalla principal o realizar más cambios en la aplicación. En caso de volver a la pantalla principal, se mostrarían los contactos seleccionados.



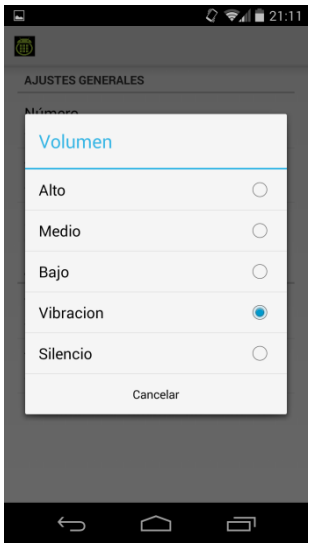
El siguiente ajuste es el fondo de la aplicación, que puede elegirse entre la galería de la aplicación o alguna imagen del teléfono. Si se selecciona la opción de fondo de la aplicación, se mostrarán cuatro fondos distintos disponibles, y se aplicarán los cambios al volver a la pantalla principal:





Al igual que con la galería de la aplicación, si se selecciona una imagen del dispositivo, se pondrá de fondo de pantalla.

El siguiente ajuste es el volumen del teléfono, en el cual se puede seleccionar una de las siguientes cinco posibilidades:



Finalmente, la última de las opciones de personalización es la opción de cambiar el tono del teléfono, como se ve en la siguiente figura: